QUADRASHAP: ϵ-EXACT SHAPLEY VALUES FOR PRODUCT GAMES IN LOGARITHMIC PARALLEL TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce QUADRASHAP, a method for ϵ-*exact* Shapley computation in *product games*, cooperative games whose coalition values factorize multiplicatively across players. Given a tolerance $\epsilon \geq 0$, our method determines a sufficient computational budget before evaluation, guaranteeing an absolute attribution error of at most ϵ for every feature in exact arithmetic. By extending to weighted sums of product games, the framework supports baseline and empirical interventional attribution across a broad class of models, including log-link regression models, Cox proportional-hazards models, odds-scale classifiers, product-kernel machines, and tree-based models. For a d-player product game, we replace the exponentially large coalition sum by a one-dimensional integral of a polynomial of degree at most $d-1$. Gauss–Legendre quadrature with $m_q = \lceil d/2 \rceil$ nodes therefore recovers exact Shapley values when $\epsilon = 0$. For given positive tolerances, we derive a computable, factor-dependent error bound that decays geometrically in m_q and yields a sufficient node budget for the prescribed tolerance. Efficient evaluation at scale is enabled by sharing computations across features and evaluating products in log-space with sign tracking, mitigating intermediate overflow and underflow. Given the quadrature rule, computing all d attributions requires $O(d\, m_q)$ work per product game, compared with $O(2^d)$ for direct coalition summation, and admits $O(\log d)$ parallel time with sufficient processors. On a survival analysis problem with $396{,}065$ features, the exact quadrature configuration computes all feature attributions in approximately five minutes of GPU evaluation per explanation, at a scale where exhaustive coalition enumeration is infeasible. Explicit error control enables a further reduction: setting $\epsilon = 10^{-6}$ selects a smaller, theoretically certified quadrature budget and reduces the mean evaluation time to 0.36 seconds per explanation. These results demonstrate scalable attribution with explicit error control and motivate application of low-latency explanations for real-time safety-critical decision support. Code is available at `https://anonymous.4open.science/r/quadrashap-28D7`.

1 INTRODUCTION

Shapley values (SVs) (Shapley, 1953) explain model predictions by assigning feature contributions according to axioms from cooperative game theory (Lundberg & Lee, 2017). Their definition averages each feature's contribution over subsets of features, called *coalitions*. Direct evaluation involves exponentially many coalitions, making exact computation expensive in general. Existing methods address this cost through model-agnostic approximation (Lundberg & Lee, 2017; Muschalik et al., 2024; Fumagalli et al., 2026) or algorithms that exploit the structure of particular model classes (Lundberg et al., 2020; Yu et al., 2022a; Chau et al., 2022; 2023; Mohammadi et al., 2026b).

We focus on a shared structure that makes a common computation possible: coalition values that factorize multiplicatively across features. In a *product game* (Dehez, 2024), the value assigned to a coalition $\mathcal{S}$ takes the form $v(\mathcal{S}) = \prod_{j \in \mathcal{S}} u_j$ where u_j is the factor associated with player (feature) j (Definition 1). By linearity, the SVs of weighted sums of these games follow from the same computation. Extending this formulation to products with factors for both present and absent features covers log-link generalized linear and additive models, Cox proportional-hazards models, logistic regression and naive Bayes on the odds scale, product-kernel machines, and decision trees (Table 1).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Exploiting this structure requires both computational efficiency and numerical stability. Polynomial interpolation can eliminate coalition enumeration, but the Vandermonde transformations used in interpolation-based Shapley methods can severely amplify rounding errors even at moderate polynomial degrees (Yu et al., 2022a; Gautschi, 1975). Separately, directly multiplying thousands of feature-wise factors can cause overflow or underflow, even when individual factors have moderate magnitudes. Computational cost remains another obstacle: polynomial-time exact evaluation can still be prohibitive with hundreds of thousands of features. A useful method should therefore address both sources of numerical error and explicitly control error when trading accuracy for computation.

We introduce QUADRASHAP, which selects a computational budget to meet a user-specified accuracy target. For a specified game and tolerance $\epsilon \geq 0$, we call a computation ϵ-*exact* if the absolute error in every feature's Shapley value is at most ϵ in exact arithmetic. Setting $\epsilon = 0$ recovers exact Shapley values, while positive tolerances can permit smaller computational budgets. The guarantee controls quadrature error: the error introduced by approximating an integral with a weighted sum of function evaluations. Error guarantees are also available for other Shapley estimators: sampling-based methods can provide finite-sample probabilistic guarantees (Maleki et al., 2013), and quadrature over permutations admits deterministic error bounds (Mitchell et al., 2022a). Our contribution is to exploit product structure to derive an explicit relationship between accuracy and computational budget. Whereas multilinear sampling methods estimate coalition averages within an integral representation (Okhrati & Lipani, 2021), we compute these averages analytically, reducing the problem to a one-dimensional polynomial integral. From the factors of this polynomial, we derive a computable quadrature-error bound that determines a sufficient node count before evaluation.

Concurrent with our work, TreeGrad-Shap (Li et al., 2026) and Quadrature-TreeSHAP (Wettenstein et al., 2026) have also proposed using Gauss–Legendre (GL) quadrature for path-dependent tree attribution. For a given node count, our tree specialization evaluates the same leaf-wise integrand at the same GL nodes. In contrast, we develop this computation for the broader class of product games, together with a computable accuracy guarantee and an evaluator designed for long products. Section A provides a detailed comparison.

Our contributions are threefold: First, we provide ϵ-exact computation with a sufficient budget determined before evaluation. By expressing the Shapley weights as integrals, we replace the exponential coalition sum for a d-player product game by a one-dimensional integral of a polynomial of degree at most $d-1$ (Section 3). GL quadrature with $m_q = \lceil d/2 \rceil$ nodes gives exact computation. For positive tolerances, we derive a geometrically decaying error bound that is computable from the game's factors and can certify smaller node counts (Proposition 3 and Section 5.1). The exactness threshold remains a fallback whenever no smaller budget is certified.

Second, we share computation across features through log-space evaluation and parallel reductions. At each quadrature node, the feature attributions depend on products that differ by only one factor. Reusing these products gives $O(d\, m_q)$ work per game, while accumulating logarithms and tracking signs mitigates intermediate overflow and underflow (Section 3.3). Combining partial sums and products in a binary tree yields $O(\log d)$ parallel time for $m_q = O(d)$, given the quadrature rule and sufficient processors (Proposition 4). At the exactness threshold $m_q = \lceil d/2 \rceil$, this evaluator uses $O(d^2)$ total work.

Third, we give a common formulation across models and attribution settings. For models expressed as weighted sums of feature-wise products, we derive the factors needed for baseline attribution, which replaces absent features with values from one reference input, and empirical interventional attribution, which averages these replacements over a background dataset. The same formulation covers neutral-factor attribution, which replaces absent multiplicative factors by one (Mohammadi et al., 2026a). Each setting reduces to weighted product evaluations, so the same algorithm and error analysis apply once the factors are constructed (Section 4).

2 PRELIMINARIES

Notation. Let $\mathcal{D} = \{1, \ldots, d\}$ index the input features, and let $2^{\mathcal{D}}$ denote the collection of all feature subsets, or coalitions. We write $\mathbf{x} \in \mathbb{R}^d$ for the input to be explained, X for a random input, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ for the training feature matrix. For $\mathcal{S} \subseteq \mathcal{D}$, $\mathbf{x}_{\mathcal{S}}$ and $\mathbf{X}_{\mathcal{S}}$ restrict the vector and matrix to

the corresponding coordinates and columns. Expectation is denoted by $\mathbb{E}$. Proofs are deferred to Appendix E and the associated technical appendices.

Shapley values. To assign feature contributions, a value function $v : 2^{\mathcal{D}} \to \mathbb{R}$ first specifies a value for each coalition. The Shapley value (Shapley, 1953) of feature i averages the change caused by adding i to a coalition, with weights determined by coalition size:

$$\phi_i(v) = \sum_{\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}} \mu(|\mathcal{S}|)\big(v(\mathcal{S} \cup \{i\}) - v(\mathcal{S})\big), \qquad \mu(s) = \frac{s!\,(d-s-1)!}{d!}. \tag{1}$$

Here, $v(\mathcal{S} \cup \{i\}) - v(\mathcal{S})$ is the *marginal contribution* of feature i, and $\mu(s)$ is its *Shapley weight*. SVs are uniquely characterized by allocating the total difference $v(\mathcal{D}) - v(\varnothing)$ (*efficiency*), treating interchangeable features equally (*symmetry*), assigning zero to features with no marginal contribution (*null player*), and respecting weighted sums of games (*linearity*) (Shapley, 1953; Merrick & Taly, 2020). Direct evaluation of equation (1) requires 2^{d-1} coalition terms per feature, motivating the use of structure in v for efficient computation.

Value functions for explainability. For a predictive model f and an input $\mathbf{x}$, the value function specifies how to evaluate the model when only the features in $\mathcal{S}$ are treated as present (Sundararajan & Najmi, 2020; Molnar, 2022). One of the early approaches averages over absent features conditional on the observed present features, giving the *observational* or *conditional* value $v^{\text{obs}}_{\mathbf{x}}(\mathcal{S}) = \mathbb{E}\big[f(X) \mid X_{\mathcal{S}} = \mathbf{x}_{\mathcal{S}}\big]$. Alternatively, drawing the absent features jointly from their marginal distribution gives the *interventional* or *marginal* value $v^{\text{int}}_{\mathbf{x}}(\mathcal{S}) = \mathbb{E}_{X_{\mathcal{D}\setminus\mathcal{S}}}\big[f(\mathbf{x}_{\mathcal{S}}, X_{\mathcal{D}\setminus\mathcal{S}})\big]$. This second choice preserves dependence among absent features but does not condition their distribution on the present features (Janzing et al., 2020; Chen et al., 2020; Heskes et al., 2020).

Concretely, we use three value functions. The *empirical interventional* value function replaces this expectation by averaging over fixed reference inputs that supply values for absent features (i.e., the *background dataset*). A single reference input gives *baseline attribution*. For multiplicative models, *neutral-factor attribution* instead replaces absent factors by 1 (Appendix D).

Prescribed accuracy: ϵ-exactness. To make the accuracy–computation trade-off explicit, we allow the user to prescribe an absolute error tolerance for every feature. For a fixed game v and $\epsilon \geq 0$, we call a computation ϵ-*exact* if its outputs satisfy, in exact arithmetic,

$$\max_{i \in \mathcal{D}} \big|\widehat{\phi}_i(v) - \phi_i(v)\big| \leq \epsilon. \tag{2}$$

Setting $\epsilon = 0$ recovers exact SVs, while positive tolerances permit approximation within the prescribed accuracy. Our guarantees control numerical integration error for the specified game; they do not include floating-point rounding or the (statistical) error from replacing a population expectation by a finite background average.

Product games. We begin with games in which each present feature contributes a fixed multiplicative factor, while each absent feature contributes a scalar 1 (Dehez, 2024).

Definition 1 (Product game)**.** *A value function $v : 2^{\mathcal{D}} \to \mathbb{R}$ is a* product game *if there exist factors $u_1, ..., u_d \in \mathbb{R}$ such that $v(\mathcal{S}) = \prod_{j \in \mathcal{S}} u_j$, with $\mathcal{S} \subseteq \mathcal{D}$ and the empty-product convention $v(\varnothing) = 1$.*

For feature attribution, these factors are obtained by evaluating feature-wise functions at the input $\mathbf{x}$; we write $u_j := u_j(x_j)$ for brevity. Adding feature i multiplies the coalition value by u_i, so its marginal contribution is $v(\mathcal{S} \cup \{i\}) - v(\mathcal{S}) = (u_i - 1) \prod_{j \in \mathcal{S}} u_j$. Consequently, equation (1) becomes

$$\phi_i(v) = (u_i - 1) \sum_{\mathcal{S} \subseteq \mathcal{D} \setminus \{i\}} \mu(|\mathcal{S}|) \prod_{j \in \mathcal{S}} u_j. \tag{3}$$

This factorization is the starting point for our integral representation. Section 4 extends the computation to weighted sums of products with separate factors for present and absent features. For models with a sum-of-products representation (Definition 2), this extension covers empirical interventional, baseline, and neutral-factor attribution.

Gauss–Legendre (GL) quadrature. Our approach replaces the coalition sum by an integral, then evaluates that integral through a weighted sum at selected points, called *quadrature nodes*. On $[0, 1]$, the m_q-node GL rule (Davis & Rabinowitz, 1984) takes the form $\int_0^1 g(t)\,dt \;\approx\; \sum_{q=1}^{m_q} \omega_q\, g(\tau_q)$,

where $\tau_q \in (0,1)$ and $\omega_q > 0$ are the nodes and weights. They depend only on m_q (number of quadrature nodes) and can be precomputed and reused (Golub & Welsch, 1969). The key property is polynomial exactness: in exact arithmetic, the rule integrates every polynomial of degree at most $2m_q - 1$ exactly (Trefethen, 2008). For the product-game integrals, Section 3 establishes the resulting exactness threshold and derives error bounds that certify smaller node counts when $\epsilon > 0$.

3 Shapley Values for Product Games via GL Quadrature

Our goal is to compute all Shapley values of a product game to a prescribed absolute accuracy $\epsilon \geq 0$, without enumerating feature coalitions. We first express the coalition sum as a one-dimensional integral, then use its product structure to select a sufficient quadrature budget. Finally, we share computations across features to efficiently compute all d Shapley values.

3.1 From Coalition Averages to a One-Dimensional Integral

The obstacle in equation (3) is the weighted sum over 2^{d-1} coalitions. Product structure lets us evaluate this average without visiting individual coalitions. To see why, fix $t \in [0,1]$ and include each feature $j \neq i$ independently with probability t. An included feature contributes u_j, while an excluded feature contributes one, giving the expected factor $(1-t) + tu_j$. Independence makes the expected product equal to the product of these expected factors. Averaging over t then recovers the Shapley weights.

Proposition 1 (Integral representation)**.** *Let $v(\mathcal{S}) = \prod_{j\in\mathcal{S}} u_j$ be a product game. For every $i \in \mathcal{D}$,*

$$\phi_i(v) = (u_i - 1) \int_0^1 \prod_{j\neq i} (1 - t + tu_j)\, dt. \tag{4}$$

The key identity is the integral representation of the Shapley weights, $\mu(s) = \int_0^1 t^s(1-t)^{d-s-1}\, dt$. Substituting it into equation (3) allows the coalition sum to be expanded as a product inside the integral. This is also the product-game specialization of Owen's multilinear representation (Owen, 1972), used for tree attribution by TreeGrad-Shap (Li et al., 2026); Section A compares the formulations. Here, the product structure makes the integrand directly evaluable, with no inner coalition sampling.

3.2 Choosing a Budget for Prescribed Accuracy

Each integrand in equation (4) is a polynomial of degree at most $d-1$. Since an m_q-node GL rule integrates polynomials of degree at most $2m_q - 1$ exactly, $\lceil d/2 \rceil$ nodes suffice for exact SVs. The practical question is whether fewer nodes can certify a positive tolerance.

To bound the error of smaller rules, we control the polynomial's magnitude on an ellipse surrounding the integration interval in the complex plane. For $\rho > 1$, let E_ρ denote the *Bernstein ellipse* with foci 0 and 1 and semi-major axis $(\rho + \rho^{-1})/4$ (Trefethen, 2008). The parameter ρ controls the ellipse's size; Definition 3 in Appendix C gives its precise construction.

Proposition 2. *For a product game v, let $(\tau_q, \omega_q)_{q=1}^{m_q}$ be GL nodes and weights on $[0,1]$. Define,*

$$\widehat{\phi}_i(v) := (u_i - 1) \sum_{q=1}^{m_q} \omega_q \prod_{j\neq i} \big(1 - \tau_q + \tau_q u_j\big). \tag{5}$$

Then:

(i) Exactness. *If $m_q \geq \lceil d/2 \rceil$, then $\widehat{\phi}_i(v) = \phi_i(v)$ for every $i \in \mathcal{D}$.*

(ii) Error bound. *For each i, let $g_i(z) := \prod_{j\neq i}(1 - z + zu_j)$ and $M_\rho := \max_{z\in E_\rho} |g_i(z)|$. Then,*

$$\big|\widehat{\phi}_i(v) - \phi_i(v)\big| \leq |u_i - 1|\, C_\rho M_\rho\, \rho^{-2m_q}, \qquad C_\rho := \frac{32}{15}\frac{1}{1-\rho^{-2}}, \quad \forall \rho > 1. \tag{6}$$

For fixed ρ, the upper bound decreases geometrically with m_q. However, it still contains the unknown maximum M_ρ. The product structure allows us to bound this quantity directly from the factors. What matters is how much the factors vary relative to their magnitudes, together with the absolute scale of the product.

Proposition 3. *Let* $M_j := \max(1, |u_j|), \Pi_i := \prod_{j \neq i} M_j, \Lambda_i := \sum_{j \neq i} \frac{|1-u_j|}{M_j}$. *Then*

$$\left|\widehat{\phi}_i(v) - \phi_i(v)\right| \leq |u_i - 1|\, \Pi_i\, B(m_q, \Lambda_i) \tag{7}$$

where $B(m, \Lambda) := \inf_{\rho > 1} C_\rho \exp\left(\frac{\Lambda}{4}(\rho + \rho^{-1} - 2)\right) \rho^{-2m}$. *For* $m \geq 1$ *and* $\Lambda \geq 0$, B *is non-increasing in* m *and non-decreasing in* Λ. *For* $\Lambda > 0$, *its minimizer is obtained by solving a scalar equation; for* $\Lambda = 0$, $B(m, 0) = 0$.

The quantity Λ_i sums the changes from the absent factor one to the present factors, each relative to its magnitude. We call it the *total relative variation*. The prefactor $|u_i - 1|\Pi_i$ sets the absolute scale of the error bound. Both quantities therefore influence the certified budget. In general, $\Lambda_i \leq 2(d-1)$; for nonnegative factors, $\Lambda_i \leq d - 1$. When many factors are close to one, the total relative variation can remain small even in high dimensions, allowing smaller budgets when the scale of the bound also permits them.

Selecting the node count. Let $m_{\text{exact}} = \lceil d/2 \rceil$. For $\epsilon = 0$, we use m_{exact}. For $\epsilon > 0$, we search for the smallest $m < m_{\text{exact}}$ satisfying $\max_{i \in \mathcal{D}} |u_i - 1|\, \Pi_i\, B(m, \Lambda_i) \leq \epsilon$. Monotonicity permits bisection over the candidate node counts. If no smaller count passes, we return m_{exact}, whose guarantee follows from polynomial exactness rather than from the analytic error bound. Thus, every prescribed tolerance has a sufficient budget. This budget is the smallest accepted by the certificate and exactness fallback, not necessarily the smallest that achieves the requested accuracy in practice. Section C gives the derivation and extends the certificate to weighted sums of games; Algorithm 2 summarizes the procedure.

The resulting computation evaluates products at quadrature nodes, without enumerating coalitions or recovering polynomial coefficients. It therefore avoids the coefficient-recovery step of interpolation-based approaches (Yu et al., 2022a; Heidari et al., 2026), which inverts an ill-conditioned Vandermonde system and can amplify numerical errors (Gautschi, 1975). Each product, however, has $d - 1$ factors, and a product of thousands of factors can overflow or underflow floating-point arithmetic even when every factor is moderate; forming it separately for each feature would also cost $O(d^2 m_q)$ work. The next subsection addresses both: it evaluates the products in log-space so that their magnitude never leaves the representable range, and shares them across features so that the work is $O(d\, m_q)$.

3.3 Shared Evaluation and Parallel Complexity

Reusing products across features. At each quadrature node, the products required by different features differ by only one factor. Computing each independently would take $O(d^2 m_q)$ work. Sharing their common factors reduces this to $O(d\, m_q)$, while log-space arithmetic mitigates the intermediate overflow and underflow that can arise when multiplying many factors.

For each node, define $T_{q,j} := (1 - \tau_q) + \tau_q u_j$, $P_q := \prod_{j=1}^{d} T_{q,j}$. When every $T_{q,j}$ is nonzero, the product needed for feature i is $P_q / T_{q,i}$. Rather than forming P_q directly, we accumulate its log-magnitude and track its sign by setting $\log |P_q| = \sum_{j=1}^{d} \log |T_{q,j}|$ and $\operatorname{sgn}(P_q) = \prod_{j=1}^{d} \operatorname{sgn}(T_{q,j})$. Removing the corresponding log-magnitude and sign gives

$$\widehat{\phi}_i(v) = (u_i - 1) \sum_{q=1}^{m_q} \omega_q \operatorname{sgn}(P_q) \operatorname{sgn}(T_{q,i}) \exp\big(\log |P_q| - \log |T_{q,i}|\big). \tag{8}$$

The shared quantities are computed once per node and reused for every feature. If all u_j are nonnegative, every $T_{q,j}$ is strictly positive because $\tau_q \in (0, 1)$, so sign tracking is unnecessary. For signed factors, some $T_{q,j}$ may vanish, making logarithms and division undefined. The division-free evaluator in Section B handles these cases by combining the products before and after each feature, with the same asymptotic complexity.

Log-space evaluation addresses intermediate numerical range; it does not eliminate rounding error or cancellation, and reconstructed products must still fit within the floating-point range. The ϵ-exactness certificate therefore remains a guarantee on quadrature error in exact arithmetic.

Parallel evaluation. The shared log-magnitudes, signs, and final weighted sums can each be computed by repeatedly combining pairs of partial results. For a sequence of length k, this *tree reduction* uses $O(k)$ total work and $O(\log k)$ parallel time (Blelloch, 1990). The reductions over features can

run simultaneously at all nodes, and the subsequent sums over quadrature nodes simultaneously for all features; since $m_q \leq \lceil d/2 \rceil$, both take $O(\log d)$ parallel time.

Proposition 4. *Let d be the number of features and m_q the number of GL nodes.*

(i) (Sequential.) *The estimator $\{\widehat{\phi}_i\}_{i=1}^d$ in equation* (8) *requires $O(d\, m_q)$ total work.*
(ii) (Parallel.) *The parallel time complexity is $O(\log d)$ with $O\big(d\, m_q/(\log d + \log m_q)\big)$ processors.*

In addition to the time complexity, memory requirements depend on how the product-game components are constructed and accumulated from ML models. Section 4 examines this issue for empirical interventional value functions and describes a blockwise implementation that avoids materializing all component-background factor tables simultaneously.

4 Multiplicative Models and Their Induced Product Games

Section 3 developed an exact, numerically stable algorithm for a single product game. We now show that models with a multiplicative representation across features induce value functions, including empirical interventional as well as baseline, that are weighted sums of product games.

Definition 2. *A predictive model $f : \mathbb{R}^d \to \mathbb{R}$ is* multiplicative *with p components if, for $c_r \in \mathbb{R}$,*

$$f(\mathbf{x}) = \sum_{r=1}^{p} c_r \prod_{j \in \mathcal{D}} u_j^{(r)}(x_j). \tag{9}$$

Each index r identifies a multiplicative component, formed as a product of per-feature terms. When $p = 1$, the model is a single product, while $p > 1$ gives a finite sum of such products. The next subsection applies this representation to several model classes, where p is one, the number of training points, or the number of rules.

The product game of Definition 2 provides the basic computation for a single multiplicative component, with absent features contributing one. To accommodate different value functions, we retain $u_j^{(r)}(x_j)$ for present features and assign each absent feature a fixed factor $\tilde{u}j^{(r)}$, independent of the coalition and its observed values. *Neutral-factor attribution* sets $\tilde{u}j^{(r)} = 1$, so each component is directly a product game; this construction has been used for product-kernel attribution (Mohammadi et al., 2026a; Gevaert & Saeys, 2024). *Baseline attribution* instead fixes a reference input $\mathbf{x}^b$ and sets $\tilde{u}j^{(r)} = u_j^{(r)}(x_j^b)$, giving $v\mathbf{x}(\mathcal{S}) = f(\mathbf{x}\mathcal{S}, \mathbf{x}^b\mathcal{D} \setminus \mathcal{S})$. When the absent factors are nonzero, each component is a scaled product game with factors $u_j^{(r)}(x_j)/\tilde{u}_j^{(r)}$. The division-free formulation below also covers zero absent factors.

The *empirical interventional* value function averages these baseline evaluations over a fixed set of reference inputs, referred to as the *background dataset*, typically drawn from the training data. The following proposition expresses this average as a weighted sum of product-form games, one for each component–background pair. Applying the single-game integral construction to each pair and combining its attributions by linearity then gives one quadrature formula for all three value functions. Baseline attribution uses one reference input, while neutral-factor attribution sets all absent factors to one; both are covered by the quadrature formulation below (Appendix D).

Proposition 5. *Let f be a multiplicative model as in equation* (9)*, and let $\mathcal{B} = \{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(n_b)}\}$ be a background dataset. Define $u_j^{(r)} = u_j^{(r)}(x_j)$ and $\tilde{u}_j^{(rb)} = u_j^{(r)}(\tilde{x}_j^{(b)})$. Then the empirical interventional value function takes the form*

$$\hat{v}_{\mathbf{x}}(\mathcal{S}) = \frac{1}{n_b} \sum_{r=1}^{p} \sum_{b=1}^{n_b} c_r \prod_{j \in \mathcal{S}} u_j^{(r)} \prod_{j \notin \mathcal{S}} \tilde{u}_j^{(rb)}. \tag{10}$$

The two sums in equation (10) do not complicate the attribution: by linearity, the SVs can be computed separately for each component–background pair and then summed with the same weights. Each pair contributes a product with one factor per feature, $u_j^{(r)}$ where the feature is present and $\tilde{u}_j^{(rb)}$ where it is absent, to which the integral argument of Proposition 1 applies unchanged, giving a division-free formula.

Table 1: Model classes with multiplicative structure: quantity explained, per-feature factor and number of components p. Notation: s_j are the smooth terms of a log-link GAM ($s_j(x_j) = \beta_j x_j$ for a GLM), $\pi(\mathbf{x}) = \Pr(Y{=}1 \mid \mathbf{x})$, h_0 the baseline hazard, $k = \prod_j k_j$ a product kernel with coefficients α_i, and rule ℓ tests the features J_ℓ with scores $q_{\ell j}$.

Model	Quantity explained	Factor $u_j^{(r)}(x_j)$	p
GLM / GAM, log link	$\mathbb{E}[Y \mid \mathbf{x}]$	$e^{s_j(x_j)}$	1
Logistic regression	odds $\pi(\mathbf{x})/(1-\pi(\mathbf{x}))$	$e^{\beta_j x_j}$	1
Naive Bayes	posterior odds	$p(x_j \mid Y{=}1)/p(x_j \mid Y{=}0)$	1
Cox proportional hazards	relative hazard $h(t \mid \mathbf{x})/h_0(t)$	$e^{\beta_j x_j}$	1
Product-kernel predictor	$f(\mathbf{x}) = \sum_i \alpha_i\, k(\mathbf{x}, \mathbf{x}^{(i)})$	$k_j(x_j, x_j^{(r)})$	n
Rule / tree ensemble	$f(\mathbf{x}) = \sum_\ell c_\ell \prod_{j \in J_\ell} q_{\ell j}(x_j)$	$q_{\ell j}(x_j)$, and 1 for $j \notin J_\ell$	number of rules

Proposition 6. *Let $\hat{v}_{\mathbf{x}}$ be as in equation* (10) *and $\{(\tau_q, \omega_q)\}_{q=1}^{m_q}$ be GL nodes and weights. Then,*

$$\widehat{\phi}_i(\hat{v}_{\mathbf{x}}) := \frac{1}{n_b} \sum_{r=1}^{p} \sum_{b=1}^{n_b} c_r \big(u_i^{(r)} - \tilde{u}_i^{(rb)}\big) \sum_{q=1}^{m_q} \omega_q \prod_{j \neq i} \big((1-\tau_q)\tilde{u}_j^{(rb)} + \tau_q u_j^{(r)}\big). \tag{11}$$

Equation (11) is equation (5) with $T_{q,j} = (1-\tau_q)\tilde{u}_j + \tau_q u_j$ in place of $(1-\tau_q) + \tau_q u_j$, so the shared log-space evaluator of Section 3.3 applies to it verbatim; where absent factors vanish, as when a rule indicator is inactive at a background point, the division-free evaluator of Section B is used. Because whole background rows are used, dependence among the absent features is preserved exactly; only the dependence between present and absent features is severed, and that by the interventional value function itself.

Time and memory complexity. Proposition 4 applies once per component–background pair, giving $O(p\, n_b\, d\, m_q)$ sequential work; the pairs are independent and their aggregation is one more associative reduction, so the parallel time is $O(\log d + \log(p\, n_b))$. For the neutral-factor and baseline value functions $n_b = 1$, and the work is $O(p\, d\, m_q)$. The node budget of Proposition 3 carries over to equation (11) (Section C). Materializing all factor tables at once would require $O(p\, n_b\, d)$ memory. Since the outer sum in equation (10) is a reduction, the pairs can be processed in blocks of size B and accumulated into the running attribution vector, which reduces peak memory to $O(Bd)$.

Model classes with multiplicative structure. Table 1 identifies the output, per-feature factors, and number of components for common model families, allowing Proposition 6 to be applied directly. Constant prefactors, including intercept and prior-odds terms, are absorbed into c_r.

Output scale. The quantity explained is the output on which the model is multiplicative: the conditional mean of a log-link GLM or GAM, representing counts, rates, or expected costs; the posterior odds of binary logistic regression or Naive Bayes; and the relative hazard of a Cox model. For logistic regression and Naive Bayes, the choice of scale is consequential. Log-odds are additive, making interventional attribution immediate, whereas exact probability-scale Shapley computation is #P-hard in general, even under fully factorized data distributions (Van den Broeck et al., 2022). The odds scale preserves the multiplicative structure and permits exact computation through our framework. By efficiency, these SVs decompose the difference between the odds at $\mathbf{x}$ and the average background odds, providing an odds-scale counterpart to the log-odds explanations commonly used for classification (Lundberg & Lee, 2017; Lundberg et al., 2020).

Tree structure. Tree ensembles fit the same framework through their root-to-leaf rules, with node counts governed by the distinct features on each path rather than the full input dimension. The path-dependent value function, tree-specific node counts, and implementation are detailed in Section F.

5 Experiments

We evaluate QuadraSHAP along three dimensions: the node counts needed to meet a prescribed accuracy, runtime against PKeX-Shapley on synthetic and text-classification tasks, and scalability to high-dimensional Cox models. Comparisons with existing tree explainers, together with additional node-budget and interventional experiments, are reported in Section G.

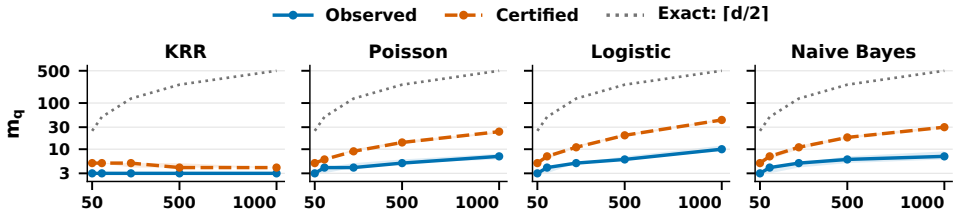

Figure 1: **Nodes needed for maximum absolute attribution error** 10^{-6} versus the number of features d. Blue: *observed*; orange: *certified*; gray: full-dimensional polynomial exactness, $\lceil d/2 \rceil$. Points are medians and bands are 10th–90th percentiles over five fits and 20 inputs per fit.

Table 2: Product-kernel benchmarks on synthetic and TF-IDF data. Median runtime (s/input) for the neutral-factor game, with node budgets set by the a priori quadrature tolerance $\epsilon = 10^{-6}$. Small text: maximum absolute attribution error across inputs and features. T/O: all inputs timed out at 300 s.

	Synthetic (varying d)					TF-IDF text ($d = 5{,}000$)				
	50	500	1k	2k	5k	imdb	sms	sst2	RT	emotion
PKEX-SHAPLEY (CPU, exact)	0.171	17.5	79.4	T/O	T/O	T/O	T/O	T/O	T/O	T/O
QUADRASHAP (CPU), ϵ=10^{-6}	0.002	0.038	0.055	0.119	0.296	7.98	1.54	23.8	4.14	6.07
error $\leq$	1.8e−10	3.4e−10	1.4e−10	5.1e−11	2.3e−11	4.3e−10	3.9e−10	1.9e−11	1.3e−10	9.2e−11
QUADRASHAP (GPU), ϵ=10^{-6}	**0.001**	**0.013**	**0.019**	**0.036**	**0.082**	**2.12**	**0.385**	**4.93**	**1.23**	**1.35**
error $\leq$	1.4e−9	1.7e−8	3.4e−8	1.9e−8	8.1e−9	4.3e−10	3.9e−10	2.0e−11	1.4e−10	9.5e−11

5.1 CERTIFIED AND OBSERVED QUADRATURE BUDGETS

We train RBF kernel ridge regression (KRR), Poisson regression, logistic regression, and Gaussian Naive Bayes (NB) on synthetic Gaussian features with $d \in \{50, 100, 250, 500, 1000\}$, of which 30% are informative (Sections G.2.1 and G.2.2). For each model and dimension, five independent fits and 20 held-out inputs per fit give 100 explanations. We explain the KRR output, the Poisson mean, and the logistic and NB odds. For every input, we first compute the exact Shapley vector with $m_q = \lceil d/2 \rceil$ nodes; the model and the input then stay fixed while m_q is reduced, and the error at each m_q is the largest absolute difference from the exact vector over the input's features. We compare two node counts. The *observed* count is the smallest tested m_q whose error is at most $\epsilon = 10^{-6}$; it is known only in hindsight, because it uses the exact vector. The *certified* count is the smallest m_q that guarantees the given accuracy.

Figure 1 plots the certified and observed node counts against d. The certificate held for all explanations, with a worst error of 7.9×10^{-9} at the certified count. At $d = 1000$ the median certified counts are 4 (KRR), 24 (Poisson), 43 (logistic), and 30 (NB), against an exactness threshold of 500; the largest count for the given accuracy over all inputs was 47. The count that would have sufficed in hindsight is smaller, at most 14 nodes for any input at $d = 1000$. Counts grow with d for Poisson, logistic, and NB but stay at 4–5 for KRR at every d, so the feature count alone does not determine the nodes required for a given accuracy (Proposition 3). Appendix G.3 provides more empirical insights on the other factors that do, including signal strength, the value function, and the variation of the factors u_j.

5.2 PRODUCT-KERNEL METHODS BENCHMARK

We compare QUADRASHAP with PKEX-SHAPLEY (Mohammadi et al., 2026a), the exact baseline for product-kernel models, on the same neutral-factor game $v_x(S) = \sum_r \alpha_r \prod_{j \in S} \exp[-\gamma(x_j - z_{rj})^2]$ (Section D); the comparison with sampling estimators, which requires a background set and hence a different game, is in Section G.4. Table 2 reports QUADRASHAP at the budget certified for $\epsilon = 10^{-6}$ (five to six nodes) on the CPU, where PKEX-SHAPLEY runs, and on the GPU. Times are median seconds per explained input; errors are the worst over the inputs.

Synthetic data. For each $d \in \{50, 500, 1000, 2000, 5000\}$ we generate 1,000 regression samples (Section G.2.1), standardize them on a shuffled 80/20 split, fit RBF KRR, and explain 50 fixed held-out samples. On the same CPU, QUADRASHAP takes 2 ms at d=50 and 0.3 s at d=5000, while PKEX-SHAPLEY takes 0.17 s, 17.5 s, and 79.4 s at $d = 50$, 500, and 1000, growing quadratically with d as its recursion predicts, and returns no explanation within the 300 s limit at $d \geq 2000$. On the GPU, QUADRASHAP needs 1–82 ms.

Table 3: Cox explanations on a GPU with 30 fixed training backgrounds and five held-out patients per cohort. $\epsilon = 0$ denotes full-degree quadrature. Δ is the maximum absolute feature-wise discrepancy from the full-degree output across patients and repetitions. Times are mean $\pm$ sample SD across patient medians of three warm calls (one call per patient for full degree).

	GSE24080 (d=54,675)			TCGA LGG (d=396,065)		
ϵ	Nodes	Δ	Time (s)	Nodes	Δ	Time (s)
0	27,338	—	4.81 ± 0.11	198,033	—	311.10 ± 0.76
10^{-1}	44–53	4.51×10^{-10}	0.06 ± 0.00	37–49	1.75×10^{-7}	0.35 ± 0.01
10^{-3}	46–54	4.48×10^{-10}	0.06 ± 0.00	39–51	1.75×10^{-7}	0.36 ± 0.00
10^{-6}	48–57	4.44×10^{-10}	0.06 ± 0.01	41–53	1.75×10^{-7}	0.36 ± 0.01

Text classification. We train an RBF SVC on each of five corpora, `imdb`, `rotten tomatoes` (RT), `sst2`, `sms spam` and `emotion` (Section G.2.3), on 5,000 TF-IDF features, and explain 20 complete held-out texts per corpus. QUADRASHAP completes every text in 1.5–24 s on the CPU and in 0.4–4.9 s on the GPU; PKEX-SHAPLEY returns no attribution within 300 s for any of the 100 texts, so its per-text time is above 300 s rather than equal to it and its error is unavailable.

5.3 HIGH-DIMENSIONAL COX SURVIVAL MODELS

To assess whether QUADRASHAP can deliver accurate explanations at extreme dimensionality, we explain ridge-penalized Cox models of overall survival (Section G.2.4) trained on GSE24080 gene expression (MAQC Consortium, 2010) and TCGA lower-grade glioma methylation (Cancer Genome Atlas Research Network, 2015), with $(n_{\text{train}}, d) = (339, 54675)$ and $(383, 396065)$, respectively. Each cohort uses 30 fixed, uniformly weighted training-background patients and five held-out patients. At a quadrature tolerance of $\epsilon = 10^{-6}$, automatic node selection requires only 48–57 and 41–53 nodes, compared with 27,338 and 198,033 for full-degree quadrature. On an NVIDIA RTX 5090, mean evaluation times are 0.06 and 0.36 seconds, versus 4.81 and 311.10 seconds for full-degree evaluation (Table 3). The approximate timings average the median of three warm calls per patient and include automatic node selection; all timings exclude quadrature-rule construction and warmup. Maximum absolute feature-wise discrepancies from the corresponding full-degree outputs are 4.44×10^{-10} and 1.75×10^{-7}, below the requested tolerance for every patient. Δ is the same at all three tolerances, so it is consistent with the rounding level of the full-degree reference rather than the quadrature error, which is already below that level at $\epsilon = 10^{-1}$.

These results highlight why reducing exponential coalition enumeration to polynomial work is not sufficient by itself at this scale. Existing exact methods with $O(d^2)$ complexity substantially improve on $O(2^d)$ enumeration, yet quadratic scaling can remain costly with hundreds of thousands of features, as our own full-degree timings also illustrate. Standard Monte Carlo coalition sampling offers probabilistic error control, but conventional bounds give sufficient sample budgets that grow as ϵ^{-2} for fixed confidence and marginal-contribution range. Exploiting product structure instead allows QUADRASHAP to select a sufficient budget before evaluation through a geometrically decaying, deterministic quadrature-error bound.

6 CONCLUSION

QUADRASHAP combines efficient Shapley computation with explicit error control: a factor-based certificate selects a sufficient quadrature budget before evaluation, guaranteeing the requested attribution accuracy in exact arithmetic. This budget can be substantially smaller than that required for polynomial exactness, making low-latency explanations possible even in very high dimensions. For a Cox model with 396,065 features, setting $\epsilon = 10^{-6}$ reduced mean GPU evaluation time from 311 seconds for the exact quadrature configuration to 0.36 seconds per explanation, excluding quadrature-rule construction. These results bring large-scale attribution closer to interactive use, while retaining a principled connection between computational effort and accuracy. Floating-point rounding and background estimation remain separate error sources. Future work includes finite-precision error guarantees and evaluating end-to-end latency in interactive explanation workflows.

USE OF LARGE LANGUAGE MODELS

We used LLM-based tools to assist with key steps in proving mathematical claims, implementing methods, and interpreting experimental results. The authors take full responsibility for the final content.

REFERENCES

Tiago A. Almeida, José María Gómez Hidalgo, and Akebo Yamakami. Contributions to the study of SMS spam filtering: New collection and results. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pp. 259–262. ACM, 2011. doi: 10.1145/2034691.2034742.

Guy E. Blelloch. Prefix sums and their applications. Technical report, Carnegie Mellon University, 1990.

Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21 (2):201–206, 1974.

Cancer Genome Atlas Research Network. Comprehensive, integrative genomic analysis of diffuse lower-grade gliomas. *New England Journal of Medicine*, 372(26):2481–2498, 2015.

Siu Lun Chau, Robert Hu, Javier Gonzalez, and Dino Sejdinovic. RKHS-SHAP: Shapley values for kernel methods. *Advances in Neural Information Processing Systems*, 35:13050–13063, 2022.

Siu Lun Chau, Krikamol Muandet, and Dino Sejdinovic. Explaining the uncertain: Stochastic Shapley values for Gaussian process models. *Advances in Neural Information Processing Systems*, 36: 50769–50795, 2023.

Hugh Chen, Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. True to the model or true to the data? *arXiv preprint arXiv:2006.16234*, 2020.

Philip J. Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Academic Press, 2nd edition, 1984.

Pierre Dehez. Cooperative product games. *International Game Theory Review*, 26(01):2350016, 2024.

Fabian Fumagalli, R Teal Witter, and Christopher Musco. PolySHAP: Extending KernelSHAP with interaction-informed polynomial regression. *arXiv preprint arXiv:2601.18608*, 2026.

Walter Gautschi. Norm estimates for inverses of Vandermonde matrices. *Numerische Mathematik*, 23(4):337–347, 1975.

Arne Gevaert and Yvan Saeys. Unifying attribution-based explanations using functional decomposition. *arXiv preprint arXiv:2412.13623*, 2024.

Gene H. Golub and John H. Welsch. Calculation of Gauss quadrature rules. *Mathematics of Computation*, 23(106):221–230, 1969.

Farzaneh Heidari, Chao Li, and Guillaume Rabusseau. Tractable Shapley values and interactions via tensor networks. In *Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2026.

Tom Heskes, Enzo Sijben, Ioana Bucur, and Tom Claassen. Causal Shapley values: Exploiting causal knowledge to explain individual predictions of complex models. *Advances in Neural Information Processing Systems*, 33, 2020.

Dominik Janzing, Lenon Minorics, and Patrick Blöbaum. Feature relevance quantification in explainable AI: A causal problem. *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, 2020.

Weida Li, Yaoliang Yu, and Bryan Kian Hsiang Low. TreeGrad-Ranker: Feature ranking via O(l)-time gradients for decision trees. In *International Conference on Learning Representations*, 2026.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2020.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Sasan Maleki, Long Tran-Thanh, Greg Hines, Talal Rahwan, and Alex Rogers. Bounding the estimation error of sampling-based Shapley value approximation. *arXiv preprint arXiv:1306.4265*, 2013. doi: 10.48550/arXiv.1306.4265. URL https://arxiv.org/abs/1306.4265.

MAQC Consortium. The MicroArray Quality Control (MAQC)-II study of common practices for the development and validation of microarray-based predictive models. *Nature Biotechnology*, 28(8): 827–838, 2010.

Luke Merrick and Ankur Taly. The explanation game: Explaining machine learning models using Shapley values. *arXiv preprint arXiv:2009.05500*, 2020.

Rory Mitchell, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. Sampling permutations for Shapley value estimation. *Journal of Machine Learning Research*, 23(43):1–46, 2022a. URL https://jmlr.org/papers/v23/21-0439.html.

Rory Mitchell, Eibe Frank, and Geoffrey Holmes. GPUTreeShap: Massively parallel exact calculation of SHAP scores for tree ensembles. *PeerJ Computer Science*, 8:e880, 2022b.

Majid Mohammadi, Siu Lun Chau, and Krikamol Muandet. Amortized linear-time exact shapley value for product-kernel methods. 2026a. URL https://arxiv.org/abs/2505.16516.

Majid Mohammadi, Krikamol Muandet, Ilaria Tiddi, Annette Ten Teije, and Siu Lun Chau. Exact Shapley attributions in quadratic-time for fANOVA Gaussian processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24414–24421, 2026b.

Christoph Molnar. *Interpretable Machine Learning*. Lulu.com, 2022. URL https://christophm.github.io/interpretable-ml-book/.

Maximilian Muschalik, Hubert Baniecki, Fabian Fumagalli, Patrick Kolpaczki, Barbara Hammer, and Eyke Hüllermeier. shapiq: Shapley interactions for machine learning. *Advances in Neural Information Processing Systems*, 37:130324–130357, 2024.

Ramin Okhrati and Aldo Lipani. A multilinear sampling algorithm to estimate Shapley values. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 7992–7999. IEEE, 2021. doi: 10.1109/ICPR48806.2021.9412511. URL https://doi.org/10.1109/ICPR48806.2021.9412511.

Frank W. J. Olver, Daniel W. Lozier, Ronald F. Boisvert, and Charles W. Clark (eds.). *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.

Guillermo Owen. Multilinear extensions of games. *Management Science*, 18(5-part-2):64–79, 1972.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*, 2005.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. CARER: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3687–3697, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1404. URL https://www.aclweb.org/anthology/D18-1404.

Lloyd S. Shapley. A value for n-person games. In *Contributions to the Theory of Games*. Princeton University Press, 1953.

Mukund Sundararajan and Amir Najmi. The many Shapley values for model explanation. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw–Curtis? *SIAM Review*, 50(1):67–87, 2008.

Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2013.

Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of SHAP explanations. *Journal of Artificial Intelligence Research*, 74, 2022. Preliminary version appeared at AAAI 2021.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

Ron Wettenstein, Rory Mitchell, and Peng Yu. Quadrature-TreeSHAP: Depth-independent TreeSHAP and Shapley interactions. *arXiv preprint arXiv:2605.04497*, 2026.

Jilei Yang. Fast TreeSHAP: Accelerating SHAP value computation for trees. *arXiv preprint arXiv:2109.09847*, 2021.

Peng Yu, Albert Bifet, Jesse Read, Chao Xu, et al. Linear TreeSHAP. In *Advances in Neural Information Processing Systems*, 2022a.

Peng Yu, Albert Bifet, Jesse Read, Chao Xu, et al. Linear TreeSHAP: reference implementation. `https://github.com/yupbank/linear_tree_shap`, 2022b. Accessed: 2026-04-12, commit: 9deb5568.

A Relation to Quadrature-Based Tree Explainers

Two recent methods evaluate Shapley values of decision trees by Gauss–Legendre quadrature. We summarize both, state what the three approaches share, and then list where they differ. Table 4 condenses the comparison; the tree instantiation itself is given in Section F and its CPU and GPU benchmarks in Section G.5.

TreeGrad-Shap. Li et al. (2026) start from Owen's multilinear extension $\bar{f}_{\mathbf{x}}$ of the tree game (Owen, 1972) and the identity $\phi_i = \int_0^1 \partial_i \bar{f}_{\mathbf{x}}(t\mathbf{1})\, dt$. Along the diagonal $t\mathbf{1}$ the gradient is a polynomial of degree at most $D-1$ for a tree of depth D, so a Gauss–Legendre rule with $\lceil D/2 \rceil$ nodes evaluates the integral exactly, in $O(LD)$ time for a tree with L leaves; the per-leaf polynomial is accumulated along the root-to-leaf path as a product of factors of the form $(1-t)+t\gamma_e$, with the factor of an ancestor divided out when a feature is split on repeatedly. The method is presented as a by-product of the paper's main contribution, feature ranking by gradient optimization, and also yields weighted Banzhaf and Beta-Shapley values. It is presented for single decision trees under the path-dependent (cover-based) value function of TreeSHAP (Lundberg et al., 2020); ensembles follow by linearity. Numerical stability relative to Linear TreeSHAP is a headline claim, supported by experiments; the authors note that the path products can underflow for deep trees and leave this unaddressed.

Quadrature-TreeSHAP. Concurrently with the present work, Wettenstein et al. (2026) reformulate path-dependent TreeSHAP as integrals over $p \in [0,1]$ of "weighted Banzhaf polynomials": for a leaf v with leaf weight $R^v_\varnothing$ (leaf value times the product of its edge probabilities) and distinct path features $M(v)$, $R^v_\varnothing \prod_{j\in\mathcal{S}}(q_j-1)\prod_{j\in M(v)\setminus\mathcal{S}}\big((1-p)+p\,q_j\big)$, which for $|\mathcal{S}|=1$ is the integrand of Proposition 1 on the leaf's product game, and for $|\mathcal{S}|>1$ yields Shapley interaction indices of any order. The integrals are evaluated by Gauss–Legendre quadrature. The paper's central observation is empirical: although exactness requires $\lceil(\eta-|\mathcal{S}|+1)/2\rceil$ nodes, where η is the largest number of distinct features on a root-to-leaf path, a fixed budget of eight nodes reaches the numerical noise floor of the single-precision output in all their experiments, so runtime becomes practically independent of tree depth. The C++ CPU implementation has been merged into XGBoost as its TreeSHAP backend; a GPU implementation is reported separately. The method is confined to tree ensembles and the path-dependent value function.

What the three approaches share. The integral representation is the same classical identity in all three cases. For a product game the multilinear extension is $\bar{v}(\mathbf{z}) = \prod_j\big((1-z_j)+z_j u_j\big)$, so Owen's identity yields the integral representation of Proposition 1 directly; our derivation via the Beta representation of the Shapley weights arrives at the same formula. All three methods then apply Gauss–Legendre quadrature to a polynomial integrand and obtain exactness once the node count exceeds half the degree. For path-dependent trees in particular, the three methods evaluate the same per-leaf polynomial at the same nodes, at the same $O(L\,m_q)$ cost per tree; the tree instantiation of QuadraSHAP should be read in that light.

Where they differ. The differences lie in the object to which the identity is applied and in the regime the resulting algorithm is built for.

Level of generality. Both tree methods apply the identity to the tree game and are tied to the path-dependent value function. QuadraSHAP applies it to the product game as an abstract object and shows in Section 4 that the empirical interventional and baseline value function of any model with a sum-of-products representation is an exact weighted sum of such games. This is what carries the rule to product-kernel machines, log-link GLMs and GAMs, Cox models, odds-scale classifiers and rule ensembles, none of which is reachable from the tree formulation.

Polynomial degree. For a tree the degree is bounded by the number of distinct features on a path, rarely more than a few tens. For the product games of Section 4 it is d_{eff}, the number of features on which the explained instance and the background differ, which for kernels and GLMs is essentially the number of features and routinely reaches thousands. Exactness then costs $O(d^2)$, so the approximation regime with provable convergence rate is a necessity rather than an optimization.

Approximation guarantee. TreeGrad-Shap uses the exact node count only. Quadrature-TreeSHAP fixes eight nodes on empirical grounds relative to a single-precision noise floor of about 10^{-7}; its

only theoretical statement about the node count is the exactness threshold. Proposition 2 gives a geometric error bound below the threshold, and Proposition 3 turns it into a node count computed from the factors before any evaluation.

Dynamic range. A product of thousands of factors can leave the range of double precision. TreeGrad-Shap acknowledges underflow for deep trees; neither tree method handles it. QUADRASHAP evaluates all products in log-space with explicit sign tracking (Section 3.3), which is what allows the high-dimensional benchmarks of Section 5.

Parallel structure. QUADRASHAP states the computation as associative reductions and obtains $O(\log d)$ parallel time (Proposition 4). Quadrature-TreeSHAP provides SIMD-vectorized CPU code and a GPU implementation but does not analyze parallel time; TreeGrad-Shap does not discuss parallelism.

Interactions. Quadrature-TreeSHAP computes Shapley interaction indices of any order for trees. The identity extends to product games in the same way, but we do not implement or evaluate interaction indices in this paper.

In summary, on path-dependent trees the first-order specializations of the three methods evaluate the same leaf-wise quadrature integrand. Section G.5 benchmarks QUADRASHAP on CPU against the established non-quadrature tree explainers and on GPU against the XGBoost implementation of Quadrature-TreeSHAP and against GPUTreeSHAP, with the CPU implementation of TreeGrad-Shap compared separately; as discussed there, the GPU differences reflect implementation and call organization rather than a different algorithm. The contribution of QUADRASHAP is the model-independent formulation, the model classes it brings within reach, and the approximation, range, and parallelism results needed once the polynomial degree is the number of features rather than the depth of a tree.

Table 4: Quadrature-based Shapley methods compared. D is tree depth, η the largest number of distinct features on a root-to-leaf path, and d_{eff} the number of features on which present and absent factors differ.

	TREEGRAD-SHAP (Li et al., 2026)	Quadrature-TreeSHAP (Wettenstein et al., 2026)	QUADRASHAP (this paper)
Object of the identity	tree game (multilinear extension)	tree game (weighted Banzhaf polynomial)	product game, stated independently of any model
Value functions	path-dependent tree	path-dependent tree	empirical interventional for sum-of-products models; path-dependent tree; kernel-specific
Model scope	decision trees (ensembles by linearity)	tree ensembles	GLM/GAM (log link), Cox, odds-scale classifiers, product kernels, rule ensembles, trees
Polynomial degree	$\leq$ tree depth D	$\leq$ distinct path features η	d_{eff}, up to the number of features
Exact node count	$\lceil D/2 \rceil$	$\lceil (\eta - \lvert\mathcal{S}\rvert + 1)/2 \rceil$	$\lceil d_{\text{eff}}/2 \rceil$
Sub-threshold regime	not considered	fixed 8 nodes, empirical (single-precision floor)	geometric bound (Proposition 2); budget certified a priori from the factors (Proposition 3)
Interaction indices	no	any order	identity stated; not implemented
Dynamic range	underflow noted, unhandled	not addressed	log-space with sign tracking
Parallelism	not discussed	SIMD CPU, GPU implementation	$O(\log d)$ parallel time via associative reductions

Algorithm 1 Division-free evaluation of equation (5) by prefix–suffix scans

Require: factors $u_1, \dots, u_d$; Gauss–Legendre nodes and weights $\{(\tau_q, \omega_q)\}_{q=1}^{m_q}$
Ensure: $\widehat{\phi}_1, \dots, \widehat{\phi}_d$

$\widehat{\phi}_i \leftarrow 0$ for all i
for $q = 1$ **to** m_q **do** ▷ independent across q
  $T_j \leftarrow (1 - \tau_q) + \tau_q u_j$ for all j ▷ parallel over j
  $a_j \leftarrow (\log|T_j|,\ \mathrm{sgn}(T_j),\ 0)$ if $T_j \neq 0$, else $a_j \leftarrow (0, 1, 1)$, for all j
  $A^{<} \leftarrow \textsc{ExclusivePrefixScan}\big((a_j)_{j=1}^{d}; \oplus\big)$ ▷ $O(d)$ work, $O(\log d)$ parallel time
  $A^{>} \leftarrow \textsc{ExclusiveSuffixScan}\big((a_j)_{j=1}^{d}; \oplus\big)$
  for $i = 1$ **to** d **do** ▷ parallel over i
    $(\ell, s, z) \leftarrow A_i^{<} \oplus A_i^{>}$
    $\widehat{\phi}_i \leftarrow \widehat{\phi}_i + \omega_q\,(u_i - 1)\,[z = 0]\,s\,e^{\ell}$
  end for
end for

B A Division-Free Variant via Prefix–Suffix Scans

The evaluator of Section 3.3 forms the full product P_q once and recovers every leave-one-out product by a division, which is what forces the separate treatment of vanishing factors. The leave-one-out products can instead be assembled directly from an *exclusive prefix* and an *exclusive suffix* product, $\prod_{j \neq i} T_{q,j} = \big(\prod_{j<i} T_{q,j}\big)\big(\prod_{j>i} T_{q,j}\big)$, which requires neither a division nor a special case. To carry this out in log-space with zeros allowed, attach to each factor the triple

$$a_{q,j} := \begin{cases} \big(\log|T_{q,j}|,\ \mathrm{sgn}(T_{q,j}),\ 0\big), & T_{q,j} \neq 0, \\ \big(0,\ 1,\ 1\big), & T_{q,j} = 0, \end{cases} \qquad (\ell_1, s_1, z_1) \oplus (\ell_2, s_2, z_2) := (\ell_1 + \ell_2,\ s_1 s_2,\ z_1 + z_2),$$

so that a triple records the log-magnitude and sign of the product of its non-zero factors together with the number of zero factors it contains. The operation $\oplus$ is associative and commutative with identity $(0, 1, 0)$, and a triple (ℓ, s, z) decodes to the product $[z = 0]\,s\,e^{\ell}$. Writing $A_{q,i}^{<} := a_{q,1} \oplus \cdots \oplus a_{q,i-1}$ and $A_{q,i}^{>} := a_{q,i+1} \oplus \cdots \oplus a_{q,d}$ for the exclusive prefix and suffix reductions, the estimator equation (5) becomes

$$\widehat{\phi}_i(v) \;=\; (u_i - 1) \sum_{q=1}^{m_q} \omega_q\, \mathrm{dec}\big(A_{q,i}^{<} \oplus A_{q,i}^{>}\big), \qquad \mathrm{dec}(\ell, s, z) := [z = 0]\,s\,e^{\ell}. \tag{12}$$

Algorithm 1 states the procedure. All $A_{q,\cdot}^{<}$ are produced by one exclusive prefix scan of $(a_{q,j})_j$ under $\oplus$, and all $A_{q,\cdot}^{>}$ by one exclusive suffix scan, each in $O(d)$ work and $O(\log d)$ parallel time (Blelloch, 1990); the combine and decode steps are $O(1)$ per pair (i, q). Proposition 4 therefore holds verbatim for this variant. Sequentially it makes three passes over the d factors per node instead of two, a constant-factor overhead, and it performs no divisions. The same construction applies unchanged to the factors $(1 - \tau_q)\tilde{u}_j + \tau_q u_j$ of Section 4.

Comparison with the shared-product evaluator. The two evaluators compute the same quantity whenever no factor vanishes, and their asymptotic work and parallel time coincide. They differ in three respects. The shared-product form of Section 3.3 uses the scan primitive only as a reduction and needs one division per pair (i, q); the prefix–suffix form uses two genuine scans and no division. The shared-product form must detect $\mathrm{sgn}(P_q) = 0$ and recompute the affected leave-one-out products separately, whereas the prefix–suffix form treats zero factors through the counter z with no change of control flow, which matters when many components have vanishing absent factors. Finally, per node and feature the shared-product form exponentiates a difference of two accumulated logarithms, while the prefix–suffix form exponentiates a single accumulated sum; the rounding behavior of the two is the same up to that last operation.

C Choosing the Number of Nodes for a Prescribed Accuracy

Part (ii) of Proposition 2 bounds the quadrature error below the exactness threshold by an explicit multiple of ρ^{-2m_q}, but it does not yet tell a user how many nodes to take. The obstacle is the factor

$\max_{z\in E_\rho}|g_i(z)|$: it is the largest modulus of a polynomial of degree $d-1$ over an ellipse in the complex plane, it has no closed form, it depends on the free parameter ρ, and evaluating it numerically for each candidate ρ would cost more than the quadrature it is meant to certify. Turning equation (6) into a node count therefore requires an upper bound on this maximum that is explicit, is computed from the factor table in $O(d)$ operations, and holds for every $\rho > 1$ so that ρ can then be chosen optimally. This appendix provides such a bound, which is Proposition 3 of the main text, and follows it through to a procedure for the value functions of Section 4. The maximum is bounded by a single number computed from the factor table, the total relative variation Λ_i of equation (18) (Section C.1); the best ρ is the root of a scalar equation (Section C.2); and the bound is extended to the weighted sums of games of Section 4, where Algorithm 2 states the resulting procedure (Section C.3). The result is a node count, computed before any quadrature is run, that certifies the accuracy of all d attributions at once.

Notation. Throughout, $u_j \in \mathbb{R}$ is the present factor of feature j and $\tilde{u}_j \in \mathbb{R}$ its absent factor: the product game of Definition 1 is the case $\tilde{u}_j = 1$, and the component–background games of equation (10) have $\tilde{u}_j = \tilde{u}_j^{(rb)}$. For each feature j we write the linear factor, and for each i the leave-one-out integrand, as

$$T_j(t) \;:=\; (1-t)\,\tilde{u}_j + t\,u_j, \qquad g_i(t) \;:=\; \prod_{j\neq i} T_j(t), \qquad t\in[0,1]. \tag{13}$$

T_j interpolates linearly between the absent factor at $t=0$ and the present factor at $t=1$; its value $T_j(\tau_q)$ at a node is the factor $T_{q,j}$ when $\tilde{u}_j = 1$, and g_i is the integrand g_i of Proposition 2(ii), written for general absent factors as in equation (11). By Proposition 1 and the proof of Proposition 6, the exact and the quadrature Shapley values of one game are

$$\phi_i = (u_i-\tilde{u}_i)\int_0^1 g_i(t)\,dt, \qquad \widehat{\phi}_i = (u_i-\tilde{u}_i)\sum_{q=1}^{m}\omega_q\, g_i(\tau_q), \tag{14}$$

so the error in ϕ_i is $|u_i-\tilde{u}_i|$ times the quadrature error of g_i. We work with general absent factors because that is the setting of Section 4; every statement below specializes to the product game of Proposition 2 by setting $\tilde{u}_j = 1$ for all j, in which case $|u_i-\tilde{u}_i| = |u_i-1|$ and equation (16) below is exactly equation (6). We write m for the number of nodes m_q.

Definition 3 (Bernstein ellipse for $[0,1]$)**.** *For $\rho > 1$, let E_ρ be the image of the circle $\{|z| = \rho\}$ under $z \mapsto \frac{1}{2} + \frac{1}{4}\big(z + z^{-1}\big)$. It is the ellipse with foci 0 and 1, semi-major axis $(\rho+\rho^{-1})/4$ and semi-minor axis $(\rho-\rho^{-1})/4$. It crosses the real axis at $-\delta$ and at $1+\delta$, where*

$$\delta(\rho) \;:=\; \frac{\rho+\rho^{-1}-2}{4} \;=\; \sinh^2\!\Big(\frac{\ln\rho}{2}\Big) \tag{15}$$

is the distance from either focus to the nearest point of the ellipse, that is, how far E_ρ extends beyond each endpoint of $[0,1]$. The ellipse collapses onto $[0,1]$ as $\rho\downarrow 1$ and grows without bound as $\rho\to\infty$.

The proof of Proposition 2(ii) in Section E uses only that g_i is a polynomial, hence analytic in the whole plane, so equation (6) holds verbatim for the general factors of equation (13): for every $m \geq 1$ and every $\rho > 1$,

$$\big|\widehat{\phi}_i - \phi_i\big| \;\leq\; |u_i-\tilde{u}_i|\,\frac{32}{15}\,\frac{M_i(\rho)}{1-\rho^{-2}}\,\rho^{-2m}, \qquad M_i(\rho) := \max_{z\in E_\rho}|g_i(z)|. \tag{16}$$

This is the classical bound for Gauss–Legendre quadrature of analytic integrands (Trefethen, 2013, Thm. 19.3), transferred from $[-1,1]$ to $[0,1]$. Two features of it drive everything that follows. First, because g_i is a polynomial, the bound holds for *every* $\rho > 1$. Second, ρ is therefore a free parameter: a larger ρ makes ρ^{-2m} smaller but makes the ellipse larger and hence $M_i(\rho)$ larger. The node budget is decided by how this trade-off resolves, which requires knowing $M_i(\rho)$ as a function of ρ.

C.1 Bounding the integrand on the ellipse

The integrand is a product of linear factors, and a linear function is easy to maximize on an ellipse.

Lemma 1 (One linear factor)**.** *Let $u, \tilde{u} \in \mathbb{R}$, let $T(t) = (1-t)\,\tilde{u} + t\,u$ be a factor as in equation* (13)*, and let $M = \max(|u|, |\tilde{u}|)$. Then*

$$\max_{z \in E_\rho} |T(z)| \;=\; M + |u - \tilde{u}|\,\delta(\rho).$$

Proof. If $u = \tilde{u}$ then T is constant and both sides equal M. Otherwise $T(z) = (u-\tilde{u})(z-r)$ with the real root $r = \tilde{u}/(\tilde{u}-u)$, so $\max_{E_\rho}|T| = |u-\tilde{u}|\max_{E_\rho}|z-r|$, and it remains to find the point of E_ρ farthest from r. Writing a point of the circle $|w| = \rho$ as $w = \rho e^{\mathrm{i}\theta}$ in the defining map $z = \frac{1}{2} + \frac{1}{4}(w + w^{-1})$ of Definition 3 gives the parametrization

$$z(\theta) = \tfrac{1}{2} + a\cos\theta + \mathrm{i}\,b\sin\theta, \qquad a = \frac{\rho + \rho^{-1}}{4}, \quad b = \frac{\rho - \rho^{-1}}{4}, \qquad \theta \in [0, 2\pi),$$

with $a = \frac{1}{2} + \delta(\rho)$ by equation (15). Since r is real,

$$|z(\theta) - r|^2 = \big(\tfrac{1}{2} - r + a\cos\theta\big)^2 + b^2\sin^2\theta = (\tfrac{1}{2} - r)^2 + b^2 + 2a(\tfrac{1}{2} - r)\cos\theta + (a^2 - b^2)\cos^2\theta,$$

and $a^2 - b^2 = \frac{1}{4} > 0$, so the right side is a convex quadratic in $\cos\theta \in [-1, 1]$ and attains its maximum at an endpoint, $\cos\theta = \pm 1$. These are the two real vertices of the ellipse, $z(0) = \frac{1}{2} + a = 1 + \delta$ and $z(\pi) = \frac{1}{2} - a = -\delta$. Their distances to r are $|1 + \delta - r|$ and $|r + \delta|$: if $r \le \frac{1}{2}$ the farther vertex is $1 + \delta$, at distance $(1 - r) + \delta = |1 - r| + \delta$, and if $r \ge \frac{1}{2}$ it is $-\delta$, at distance $r + \delta = |r| + \delta$. In both cases the farther vertex extends the farther endpoint of $[0, 1]$ by δ, and hence

$$\max_{z \in E_\rho} |z - r| = \max\{|r|, |1 - r|\} + \delta.$$

Finally, $|u - \tilde{u}|\,|r| = |\tilde{u}|$ and $|u - \tilde{u}|\,|1 - r| = |u|$ by the definition of r, so multiplying by $|u - \tilde{u}|$ gives $\max_{E_\rho}|T| = \max\{|\tilde{u}|, |u|\} + |u - \tilde{u}|\,\delta = M + |u - \tilde{u}|\,\delta$. □

The lemma says that on the ellipse a factor is at most its size on the interval plus its slope times the overshoot $\delta(\rho)$, and it holds for any real factors, including a vanishing u or $\tilde{u}$. Multiplying over the $d - 1$ factors of g_i and using $1 + x \le e^x$,

$$M_i(\rho) \;\le\; \prod_{j \ne i}\big(M_j + |u_j - \tilde{u}_j|\,\delta(\rho)\big) \;\le\; \Pi_i\,\exp\big(\Lambda_i\,\delta(\rho)\big), \tag{17}$$

where

$$M_j := \max(|u_j|, |\tilde{u}_j|), \qquad \Pi_i := \prod_{j \ne i} M_j, \qquad \Lambda_i := \sum_{j \ne i} \frac{|u_j - \tilde{u}_j|}{M_j}, \tag{18}$$

with the convention that a factor with $M_j = 0$ makes the whole game vanish and is excluded. We call Λ_i the *total relative variation* of the factors: it adds up, over the features other than i, how much each factor changes between its present and its absent value, relative to the size of that factor. Each ratio in the sum lies in $[0, 2]$, and in $[0, 1]$ when u_j and $\tilde{u}_j$ do not have opposite signs, as for every model class of Section 4; hence $\Lambda_i \le 2(d-1)$ in general and $\Lambda_i \le d - 1$ in those cases. For the product game of Definition 1 ($\tilde{u}_j = 1$, with $u_j \in \mathbb{R}$ arbitrary) it is $\Lambda_i = \sum_{j \ne i}|1 - u_j| / \max(1, |u_j|)$. When in addition $0 \le u_j \le 1$, as for kernel factors, this is $\Lambda_i = \sum_{j \ne i}(1 - u_j)$, and for an RBF factor $u_j = \exp(-(x_j - x'_j)^2/2\ell^2)$ the inequality $1 - e^{-a} \le a$ gives $\Lambda_i \le \|\mathbf{x} - \mathbf{x}'\|^2/(2\ell^2)$. The only slack in equation (17) is that different factors attain their maxima at opposite vertices, and the inequality $1 + x \le e^x$.

What Λ_i controls. The ratio $|u_j - \tilde{u}_j|/M_j$ measures the change in factor j between its present and its absent value relative to the magnitude of that factor, and Lemma 1 says that this ratio is exactly the rate at which the factor can grow when the ellipse expands: on E_ρ the factor is at most $M_j\big(1 + \delta(\rho)\,|u_j - \tilde{u}_j|/M_j\big)$. When the present and absent factors are close on this relative scale, Λ_i is small and g_i grows only mildly as the ellipse expands, so a large ρ, and with it a fast decay ρ^{-2m}, is affordable. For a fixed ρ the bound equation (17) grows as $\exp(\Lambda_i\,\delta(\rho))$; after ρ is optimized in Section C.2, the certified error $B(m, \Lambda_i)$ is nondecreasing in Λ_i; in the regime $m \ll \Lambda_i$ the number of nodes needed for a tolerance η grows like $\sqrt{\Lambda_i \ln(1/\eta)}$ (Section C.2), so halving Λ_i saves roughly 30% of the nodes. Two consequences follow. First, features whose factor changes little between

present and absent, relative to its size, contribute little to Λ_i, and features whose factor does not change at all contribute nothing; when this holds for most features, Λ_i is far below $d-1$ and the node count is governed by Λ_i rather than by the degree $d-1$ of g_i. Second, the absolute scale of the bound is $|u_i - \tilde{u}_i|\,\Pi_i$; when every factor has magnitude at most one, as for distance-based kernels, likelihood factors and tree path factors, $\Pi_i \le 1$ and is typically much smaller, which lowers the certified error further, so the same tolerance is met with fewer nodes. These statements concern the bound, not the error itself: the actual error can be much smaller than $B(m, \Lambda_i)$, and the resulting budget is sufficient rather than minimal (see the remarks at the end of Section C.3).

C.2 Choosing the ellipse

Before Section C.1, the error bound equation (16) contained the unknown maximum $M_i(\rho)$, so the parameter ρ could not be chosen: making ρ larger was known to shrink ρ^{-2m}, but its effect on $M_i(\rho)$ was not quantified. Equation (17) supplies exactly this missing piece. Inserting it into equation (16) gives, for every $\rho > 1$,

$$\big|\widehat{\phi}_i - \phi_i\big| \;\le\; |u_i - \tilde{u}_i|\,\frac{32}{15}\,\frac{M_i(\rho)}{1-\rho^{-2}}\,\rho^{-2m} \;\le\; |u_i - \tilde{u}_i|\,\Pi_i\,\frac{32}{15}\,\frac{\exp\big(\Lambda_i\,\delta(\rho)\big)}{1-\rho^{-2}}\,\rho^{-2m}. \tag{19}$$

The right side is an elementary function of ρ in which the trade-off is in plain view: ρ^{-2m} decreases with ρ while $\exp(\Lambda_i\delta(\rho))$ increases, so for each pair (m, Λ_i) there is a best ρ, and it depends on the factors only through the single number Λ_i. Since equation (19) holds for every $\rho > 1$ and the prefactor $|u_i - \tilde{u}_i|\,\Pi_i$ does not involve ρ, we may take the infimum over ρ and obtain

$$\big|\widehat{\phi}_i - \phi_i\big| \;\le\; |u_i - \tilde{u}_i|\,\Pi_i\;B(m, \Lambda_i), \qquad B(m,\Lambda) := \inf_{\rho>1}\;\frac{32}{15}\,\frac{\exp\big(\Lambda\,\delta(\rho)\big)\,\rho^{-2m}}{1-\rho^{-2}}. \tag{20}$$

The rest of this subsection shows that the infimum is attained at a unique ρ, characterized by a scalar equation that is solved in a few iterations; states the resulting rule for selecting the number of nodes; and shows how the resulting budget scales with Λ.

Writing $\rho = e^s$ and using equation (15),

$$\ln B(m,\Lambda) = \inf_{s>0}\Big[\ln\tfrac{32}{15} - \ln\big(1-e^{-2s}\big) + \Lambda\sinh^2(s/2) - 2ms\Big]. \tag{21}$$

The bracket is strictly convex in s (its second derivative is $\frac{\Lambda}{2}\cosh s + 4e^{2s}/(e^{2s}-1)^2 > 0$), so it has a unique minimizer, characterized by

$$\frac{\Lambda}{2}\,\sinh s \;=\; 2m + \frac{2}{e^{2s}-1}. \tag{22}$$

The left side increases from 0 and the right side decreases from $+\infty$, so equation (22) has exactly one root $s^\star(m,\Lambda)$, found by bisection or Newton's method in a handful of iterations, and substituting it into equation (21) gives $B(m,\Lambda)$. Evaluating B for given m and Λ is therefore a scalar computation: one root find and one exponential. Two monotonicity properties will be used. B is decreasing in m, because a larger m shrinks ρ^{-2m} for every ρ; and B is nondecreasing in Λ, because $\exp(\Lambda\delta)$ increases with Λ for every fixed ρ.

Choosing the number of nodes. We can now state how the node count is selected for one game and a tolerance $\epsilon > 0$ on $|\widehat{\phi}_i - \phi_i|$. By equation (20), any m with $|u_i - \tilde{u}_i|\,\Pi_i\,B(m,\Lambda_i) \le \epsilon$ is sufficient, and since B is decreasing in m the smallest such m is well defined:

$$m_q(\epsilon) \;:=\; \min\Big\{\, m \ge 1 :\; |u_i - \tilde{u}_i|\,\Pi_i\,B(m,\Lambda_i) \le \epsilon \Big\} \wedge \Big\lceil \frac{d}{2} \Big\rceil, \tag{23}$$

where the cap $\lceil d/2 \rceil$ is the exactness threshold of Proposition 2(i), beyond which no certificate is needed. Since B is decreasing in m, the test $|u_i - \tilde{u}_i|\,\Pi_i\,B(m,\Lambda_i) \le \epsilon$ fails for all m below $m_q(\epsilon)$ and passes for all m at or above it, so $m_q(\epsilon)$ is located by bisection on the integer m:

1. Compute Π_i and Λ_i from the factor table by equation (18), in $O(d)$ operations. Set $m_{\mathrm{lo}} \leftarrow 0$ (the test is regarded as failing there) and $m_{\mathrm{hi}} \leftarrow \lceil d/2 \rceil$ (where the rule is exact, so the test is regarded as passing).

2. While $m_{\mathrm{hi}} - m_{\mathrm{lo}} > 1$: let $m \leftarrow \lfloor (m_{\mathrm{lo}} + m_{\mathrm{hi}})/2 \rfloor$; solve equation (22) for $s^\star(m, \Lambda_i)$ and evaluate $B(m, \Lambda_i)$ by equation (21); if $|u_i - \tilde{u}_i| \, \Pi_i \, B(m, \Lambda_i) \le \epsilon$ set $m_{\mathrm{hi}} \leftarrow m$, otherwise set $m_{\mathrm{lo}} \leftarrow m$.
3. Return $m_q(\epsilon) = m_{\mathrm{hi}}$.

The invariant is that the test fails at m_{lo} and passes at m_{hi}, so on exit m_{hi} is the smallest passing m, which is the minimum in equation (23); if no $m < \lceil d/2 \rceil$ passes, the procedure returns the exactness threshold. The search costs $\lceil \log_2 \lceil d/2 \rceil \rceil$ evaluations of B, each a scalar root find, and it is valid for every m: no restriction on m relative to Λ_i enters anywhere. This is the complete node-selection rule for a single game. The value functions of Section 4 are weighted sums of many games that share one node set, and there the same rule is applied once with a common Λ and a common scale in place of Λ_i and $|u_i - \tilde{u}_i|\Pi_i$; Section C.3 derives these and states the resulting procedure as Algorithm 2, whose search over m is the one above.

Scaling of the budget. The root of equation (22) has no closed form, but its asymptotics are simple. For $\sqrt{\Lambda} \lesssim m \ll \Lambda$ the root is $s^\star \approx 4m/\Lambda$, the bracket in equation (21) is approximately $-4m^2/\Lambda + \ln(\Lambda/m)$, and the certified error decays like $(\Lambda/m)\, e^{-4m^2/\Lambda}$: reaching a tolerance η costs $m \approx \frac{1}{2}\sqrt{\Lambda \ln(1/\eta)}$ nodes up to a logarithmic correction, the square root of Λ rather than the degree $d-1$. For $m \gg \Lambda$ the decay is super-geometric, $\exp\bigl(-2m \ln(8m/e\Lambda)\bigr)$, which is the regime that matters when Λ is small. Numerically, the budget of equation (23) lies within 5–18% of $\frac{1}{2}\sqrt{\Lambda \ln(1/\eta)}$ for $\Lambda \in [50, 5000]$ and $\eta \in [10^{-10}, 10^{-3}]$.

C.3 Weighted sums of games and the procedure

The value functions of Section 4 and Section D are weighted sums $v_{\mathbf{x}} = \sum_{r,b} w_{rb}\, v^{(rb)}$ of games with present factors $u_j^{(r)}$ and absent factors $\tilde{u}_j^{(rb)}$, all evaluated with the same node set. Write $M_j^{(rb)}$, $\Pi_i^{(rb)}$ and $\Lambda_i^{(rb)}$ for the quantities of equation (18) of the pair (r, b), and define the *full variation*

$$\bar{\Lambda} \;:=\; \max_{r,b} \sum_{j \in \mathcal{D}} \frac{|u_j^{(r)} - \tilde{u}_j^{(rb)}|}{M_j^{(rb)}}, \tag{24}$$

one sum per pair, so that $\Lambda_i^{(rb)} \le \bar{\Lambda}$ for every i, r and b. By linearity of ϕ_i and $\widehat{\phi}_i$, the triangle inequality, equation (20) for each pair, and the monotonicity of B in Λ,

$$\begin{aligned} \bigl|\widehat{\phi}_i(v_{\mathbf{x}}) - \phi_i(v_{\mathbf{x}})\bigr| \;&\le\; \sum_{r,b} |w_{rb}| \, \bigl|u_i^{(r)} - \tilde{u}_i^{(rb)}\bigr| \, \Pi_i^{(rb)} \, B\bigl(m, \Lambda_i^{(rb)}\bigr) \\ &\le\; B(m, \bar{\Lambda}) \sum_{r,b} |w_{rb}| \, \bigl|u_i^{(r)} - \tilde{u}_i^{(rb)}\bigr| \, \Pi_i^{(rb)} \;=:\; A_i \, B(m, \bar{\Lambda}). \end{aligned} \tag{25}$$

The absolute values $|w_{rb}|$ are needed because components of opposite sign may cancel in the true error, but the guarantee cannot rely on it. With $A_{\max} := \max_i A_i$,

$$\max_{i \in \mathcal{D}} \bigl|\widehat{\phi}_i(v_{\mathbf{x}}) - \phi_i(v_{\mathbf{x}})\bigr| \;\le\; A_{\max} \, B(m, \bar{\Lambda}). \tag{26}$$

The node count is therefore selected by the same bisection as for one game, equation (23), with $|u_i - \tilde{u}_i| \, \Pi_i$ replaced by $A_{\max}$ and Λ_i by $\bar{\Lambda}$; Algorithm 2 states it. All inputs are obtained in one pass over the factor tables, in $O(p\, n_b\, d)$ operations, the cost of building the tables themselves: each term of equation (24) is a sum, and the leave-one-out products $\Pi_i^{(rb)}$ come from the prefix–suffix scan of Section B, which needs no division and handles zero factors. No quadrature is involved.

Three remarks on what the certificate says. First, it bounds the quadrature error of the estimator of Proposition 6 for the given game and background set; the statistical error of the background average and floating-point rounding are separate matters. Second, the bound is absolute error on the scale $A_{\max}$, the largest possible marginal contribution, not relative error of the attribution, which can be much smaller through cancellation. For the empirical interventional value function $w_{rb} = c_r/n_b$, so A_i is an average over background points and does not grow with n_b. Third, the budget is sufficient rather than minimal: the common bound pairs the largest scale with the largest variation, which may belong to different pairs. When this is too conservative, the intermediate expression in equation (25),

Algorithm 2 Node budget for a prescribed accuracy

Require: factor tables $u_j^{(r)}$, $\tilde{u}_j^{(rb)}$ and weights w_{rb} of the value function; tolerance $\epsilon > 0$ on $\max_i |\widehat{\phi}_i - \phi_i|$

Ensure: a node count m such that $\max_i |\widehat{\phi}_i - \phi_i| \le \epsilon$

compute A_i for all i by equation (25) (prefix–suffix scan per pair) and $\bar{\Lambda}$ by equation (24); $A_{\max} \leftarrow \max_i A_i$ ▷ $O(p\, n_b\, d)$
if $A_{\max} = 0$ **then return** 1 ▷ all attributions vanish
end if
$\eta \leftarrow \epsilon / A_{\max}$; $\quad m_{\mathrm{lo}} \leftarrow 0$; $\quad m_{\mathrm{hi}} \leftarrow \lceil d/2 \rceil$ ▷ test fails at m_{lo}, passes at m_{hi}
while $m_{\mathrm{hi}} - m_{\mathrm{lo}} > 1$ **do**
  $m \leftarrow \lfloor (m_{\mathrm{lo}} + m_{\mathrm{hi}})/2 \rfloor$
  solve $\frac{\bar{\Lambda}}{2} \sinh s = 2m + 2/(e^{2s} - 1)$ for $s > 0$ ▷ root of equation (22)
  $B \leftarrow \frac{32}{15} \exp\big(\bar{\Lambda} \sinh^2(s/2) - 2ms\big)/(1 - e^{-2s})$ ▷ $B(m, \bar{\Lambda})$ by equation (21)
  if $B \le \eta$ **then** $m_{\mathrm{hi}} \leftarrow m$
  else $m_{\mathrm{lo}} \leftarrow m$
  end if
end while
return m_{hi} ▷ equals $\lceil d/2 \rceil$ if no smaller m passes; exact by Proposition 2(i)

with one root of equation (22) per pair and per candidate m, can be used in the test of Algorithm 2 instead, at negligible cost; and the same argument with $\|A\|_2 = (\sum_i A_i^2)^{1/2}$ in place of $A_{\max}$ bounds the ℓ_2 error of the whole attribution vector.

D Value Functions for Multiplicative Models

This appendix collects the value functions to which Proposition 6 applies and makes precise the claim of Section 4 that the neutral-factor and baseline value functions are simpler cases of the empirical interventional one: they fill in a single table of absent factors, whereas the interventional value function averages over n_b such tables, one per background point.

A common form. Let f be a multiplicative model as in equation (9). Every value function below can be written as

$$v_{\mathbf{x}}(\mathcal{S}) = \sum_{r=1}^{p} \sum_{b=1}^{n_b} w_{rb} \prod_{j \in \mathcal{S}} u_j^{(r)} \prod_{j \notin \mathcal{S}} \tilde{u}_j^{(rb)}, \qquad u_j^{(r)} = u_j^{(r)}(x_j), \tag{27}$$

for weights $w_{rb} \in \mathbb{R}$ and absent factors $\tilde{u}_j^{(rb)} \in \mathbb{R}$ that do not depend on $\mathcal{S}$. For each pair (r, b) the coalition value is a product with one factor per feature, $u_j^{(r)}$ if the feature is present and $\tilde{u}_j^{(rb)}$ if it is absent. The proof of proposition 6 in Section E uses only this structure: it treats one pair at a time, never divides by an absent factor, and combines the pairs by linearity of the Shapley value. It therefore gives, for any choice of w_{rb} and $\tilde{u}_j^{(rb)}$,

$$\phi_i(v_{\mathbf{x}}) = \sum_{r=1}^{p} \sum_{b=1}^{n_b} w_{rb} \left(u_i^{(r)} - \tilde{u}_i^{(rb)}\right) \int_0^1 \prod_{j \neq i} \big((1-t)\, \tilde{u}_j^{(rb)} + t\, u_j^{(r)}\big)\, dt, \tag{28}$$

and the m_q-point Gauss–Legendre rule evaluates every integral exactly whenever $m_q \ge \lceil d/2 \rceil$, since each integrand is a polynomial of degree at most $d - 1$ in t. The value functions differ only in how the tables $\tilde{u}^{(rb)}$ and the weights w_{rb} are filled in, and the cost of equation (28) is $O(p\, n_b\, d\, m_q)$, proportional to the number of tables.

Empirical interventional value function. Let $\mathcal{B} = \{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(n_b)}\}$ be a background dataset. The interventional value function $v_{\mathbf{x}}^{\mathrm{int}}(\mathcal{S}) = \mathbb{E}_{X_{\mathcal{D}\setminus\mathcal{S}}}[f(\mathbf{x}_{\mathcal{S}}, X_{\mathcal{D}\setminus\mathcal{S}})]$ of Section 2 is estimated by replacing the expectation with the average over $\mathcal{B}$,

$$\hat{v}_{\mathbf{x}}(\mathcal{S}) = \frac{1}{n_b} \sum_{b=1}^{n_b} f\big(\mathbf{x}_{\mathcal{S}}, \tilde{\mathbf{x}}_{\mathcal{D}\setminus\mathcal{S}}^{(b)}\big) = \frac{1}{n_b} \sum_{b=1}^{n_b} \sum_{r=1}^{p} c_r \prod_{j \in \mathcal{S}} u_j^{(r)}(x_j) \prod_{j \notin \mathcal{S}} u_j^{(r)}\big(\tilde{x}_j^{(b)}\big), \tag{29}$$

where the second equality substitutes equation (9) and uses that each factor depends on one coordinate only, so that the coordinates of the mixed point $(\mathbf{x}_\mathcal{S}, \tilde{\mathbf{x}}^{(b)}_{\mathcal{D}\setminus\mathcal{S}})$ separate. This is equation (27) with n_b tables, $\tilde{u}_j^{(rb)} = u_j^{(r)}(\tilde{x}_j^{(b)})$ and $w_{rb} = c_r/n_b$, which is proposition 5. Because whole background rows are used, the joint distribution of the absent features enters exactly; $\hat{v}_\mathbf{x}(\varnothing)$ is the mean prediction over $\mathcal{B}$, so by efficiency the attributions sum to $f(\mathbf{x}) - \frac{1}{n_b}\sum_b f(\tilde{\mathbf{x}}^{(b)})$.

Baseline value function. Fix a single reference point $\mathbf{x}^b$ and set $v_\mathbf{x}(\mathcal{S}) = f(\mathbf{x}_\mathcal{S}, \mathbf{x}^b_{\mathcal{D}\setminus\mathcal{S}})$. Repeating the computation in equation (29) with $\mathcal{B} = \{\mathbf{x}^b\}$ gives

$$v_\mathbf{x}(\mathcal{S}) = \sum_{r=1}^{p} c_r \prod_{j\in\mathcal{S}} u_j^{(r)}(x_j) \prod_{j\notin\mathcal{S}} u_j^{(r)}\big(x_j^b\big), \tag{30}$$

which is equation (27) with one table, $\tilde{u}_j^{(r)} = u_j^{(r)}(x_j^b)$ and $w_r = c_r$. The empirical interventional value function is the average of n_b baseline value functions, one per background point, and by linearity its Shapley values are the averages of the corresponding baseline Shapley values; the baseline value function is thus exactly the case $n_b = 1$ of proposition 6, at cost $O(p\,d\,m_q)$. Its attributions sum to $f(\mathbf{x}) - f(\mathbf{x}^b)$.

Neutral factor. The neutral factor sets $\tilde{u}_j^{(r)} \equiv 1$ for every component and every feature, so that an absent feature is simply dropped from each product:

$$v_\mathbf{x}(\mathcal{S}) = \sum_{r=1}^{p} c_r \prod_{j\in\mathcal{S}} u_j^{(r)}, \qquad v_\mathbf{x}(\varnothing) = \sum_{r=1}^{p} c_r. \tag{31}$$

This is a weighted sum of product games in the sense of Definition 1, with the convention $v(\varnothing) = 1$ for each of them, and it is the value function for which Section 3 was developed: with $\tilde{u}_j = 1$, equation (28) reduces to $c_r(u_i^{(r)} - 1)\int_0^1 \prod_{j\neq i}\big((1-t)+t\,u_j^{(r)}\big)\,dt$, the integral identity of Proposition 1, and equation (5) is its quadrature. For a product-kernel predictor $f(\mathbf{x}) = \sum_{i=1}^{n}\alpha_i \prod_j k_j(x_j, x_j^{(i)})$ the neutral factor gives $v_\mathbf{x}(\mathcal{S}) = \sum_i \alpha_i \prod_{j\in\mathcal{S}} k_j(x_j, x_j^{(i)})$, that is, the predictor evaluated with the kernel restricted to the present features, which is the value function used by Mohammadi et al. (2026a); its attributions sum to $f(\mathbf{x}) - \sum_i \alpha_i$. Unlike the baseline value function, the neutral factor need not correspond to any reference point, since there need not be an $\mathbf{x}^b$ with $u_j^{(r)}(x_j^b) = 1$ for all r simultaneously. Algorithmically, however, it is again the case $n_b = 1$ of Proposition 6, with the single absent-factor table filled with ones, at cost $O(p\,d\,m_q)$.

Mean absent factor. A fourth option sits between the baseline and the empirical interventional value functions. If the background distribution factorizes across features, then $\mathbb{E}\prod_{j\notin\mathcal{S}} u_j^{(r)}(X_j) = \prod_{j\notin\mathcal{S}} \mathbb{E}\,u_j^{(r)}(X_j)$, so the interventional value function is itself of the form equation (27) with a single table, $\tilde{u}_j^{(r)} = \mathbb{E}\,u_j^{(r)}(X_j)$, estimated by the per-feature averages $\frac{1}{n_b}\sum_b u_j^{(r)}(\tilde{x}_j^{(b)})$. This costs $O(p\,d\,m_q)$ after an $O(p\,n_b\,d)$ pass over the background data, against $O(p\,n_b\,d\,m_q)$ for equation (29), but it multiplies averages where equation (29) averages products: the two coincide only when the absent features are independent, and the empirical interventional value function is the one that preserves their dependence.

E Proofs

Proof of Proposition 1

Start from equation (3). The Shapley weight is a Beta integral (Olver et al., 2010, §5.12),

$$\mu(s) = \frac{s!\,(d-s-1)!}{d!} = B(s+1,\,d-s) = \int_0^1 t^s(1-t)^{d-s-1}\,dt, \qquad s = 0,\dots,d-1.$$

Substituting it into equation (3) and exchanging the finite sum with the integral,

$$\phi_i(v) = (u_i - 1)\int_0^1 \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{i\}} \prod_{j\in\mathcal{S}} t\,u_j \prod_{j\notin\mathcal{S}\cup\{i\}} (1-t)\,dt = (u_i-1)\int_0^1 \prod_{j\neq i}\big((1-t)+t\,u_j\big)\,dt,$$

Table 5: Value functions for a multiplicative model as instances of equation (27). Each is handled by proposition 6 with the stated number of absent-factor tables, at $O(p\,d\,m_q)$ work per table; $\bar{u}_j^{(r)} = \frac{1}{n_b}\sum_b u_j^{(r)}(\tilde{x}_j^{(b)})$ and $\bar{f}_{\mathcal{B}} = \frac{1}{n_b}\sum_b f(\tilde{\mathbf{x}}^{(b)})$.

Value function	Absent factor $\tilde{u}_j^{(rb)}$	Tables	$\sum_i \phi_i$
Neutral factor (Mohammadi et al., 2026a)	1	1	$f(\mathbf{x}) - \sum_r c_r$
Baseline $\mathbf{x}^b$	$u_j^{(r)}(x_j^b)$	1	$f(\mathbf{x}) - f(\mathbf{x}^b)$
Mean absent factor	$\bar{u}_j^{(r)}$	1	$f(\mathbf{x}) - \sum_r c_r \prod_j \bar{u}_j^{(r)}$
Empirical interventional	$u_j^{(r)}(\tilde{x}_j^{(b)})$	n_b	$f(\mathbf{x}) - \bar{f}_{\mathcal{B}}$

because expanding the product over the $d-1$ factors $j \neq i$ produces exactly one term per subset $\mathcal{S}$, with the factor $t\,u_j$ for $j \in \mathcal{S}$ and $(1-t)$ for $j \notin \mathcal{S}$; this is equation (4). □

PROOF OF PROPOSITION 2

Part (i): Exactness. The integrand g_i of Proposition 2(ii) is a polynomial of degree at most $d-1$ in t. An m_q-point GL rule on $[0,1]$ integrates exactly every polynomial of degree at most $2m_q-1$ (Davis & Rabinowitz, 1984). For $m_q \geq \lceil d/2 \rceil$ we have $2m_q - 1 \geq 2\lceil d/2 \rceil - 1 \geq d-1$, so the quadrature sum reproduces the exact integral:

$$\sum_{q=1}^{m_q} \omega_q\, g_i(\tau_q) \;=\; \int_0^1 g_i(t)\,dt.$$

Multiplying both sides by $(u_i - 1)$ and applying Proposition 1 gives $\widehat{\phi}_i(v) = \phi_i(v)$.

Part (ii): Approximation. With g_i as in Proposition 2(ii), equations (4) and (5) give

$$\begin{aligned}\left|\phi_i(v) - \widehat{\phi}_i(v)\right| &= \left|(u_i-1)\int_0^1 g_i(t)\,dt - (u_i-1)\sum_{q=1}^{m_q}\omega_q\, g_i(\tau_q)\right| \\ &= |u_i - 1|\left|\int_0^1 g_i(t)\,dt - \sum_{q=1}^{m_q}\omega_q\, g_i(\tau_q)\right|. \end{aligned} \tag{32}$$

Now g_i is a polynomial in t of degree at most $d-1$, hence it is entire and in particular analytic on and inside every Bernstein ellipse E_ρ, for any $\rho > 1$. Standard Gauss–Legendre quadrature error bounds for analytic functions on $[-1,1]$ (Trefethen, 2013, Thm. 19.3), transferred to $[0,1]$ via the affine change of variables $x = 2t-1$, imply

$$\left|\int_0^1 g_i(t)\,dt - \sum_{q=1}^{m_q}\omega_q\, g_i(\tau_q)\right| \leq \frac{32}{15}\,\frac{\max_{z\in E_\rho}|g_i(z)|}{(1-\rho^{-2})\,\rho^{2m_q}}.$$

The cited theorem states the bound $\frac{64}{15}\,M\rho^{-2n}/(\rho^2-1)$ for the $(n+1)$-point rule on $[-1,1]$; with $n = m_q - 1$ and the factor $\frac{1}{2}$ from the change of variables this gives the constant above. Substituting this bound into equation (32) yields

$$\left|\phi_i(v) - \widehat{\phi}_i(v)\right| \leq \frac{32}{15}\,\frac{|u_i-1|}{1-\rho^{-2}}\,\max_{z\in E_\rho}|g_i(z)|\,\rho^{-2m_q} = |u_i-1|\,C_\rho\,M_\rho\,\rho^{-2m_q},$$

which is equation (6). Since $\rho > 1$ is arbitrary, the error decays geometrically in m_q. □

PROOF OF PROPOSITION 3

Write $\delta(\rho) := (\rho + \rho^{-1} - 2)/4$, the distance by which E_ρ extends beyond each endpoint of $[0,1]$. By Lemma 1 in Section C, each linear factor of g_i satisfies $\max_{z\in E_\rho}|(1-z)+zu_j| = M_j + |1-u_j|\,\delta(\rho)$, so

$$M_\rho \;\leq\; \prod_{j\neq i} M_j\Big(1 + \frac{|1-u_j|}{M_j}\,\delta(\rho)\Big) \;\leq\; \Pi_i\, e^{\Lambda_i\,\delta(\rho)}$$

by $1 + x \le e^x$ (equation (17)). Inserting this into equation (6) gives $|\widehat{\phi}_i - \phi_i| \le |u_i - 1|\,\Pi_i\, C_\rho\, e^{\Lambda_i \delta(\rho)} \rho^{-2m_q}$ for every $\rho > 1$, and taking the infimum over ρ yields equation (7). For fixed ρ the expression is decreasing in m and nondecreasing in Λ, hence so is the infimum. With $\rho = e^s$ the exponent is strictly convex in s, so for $\Lambda_i > 0$ the infimum is attained at the unique root of the scalar equation (22), found by bisection or Newton's method. Because B is decreasing in m, the test $\max_i |u_i - 1|\Pi_i B(m, \Lambda_i) \le \epsilon$ is monotone in m, so bisection over $\{1, \dots, \lceil d/2 \rceil\}$ returns the smallest m_q that passes; at $m_q = \lceil d/2 \rceil$ the rule is exact by part (i) of Proposition 2. □

PROOF OF PROPOSITION 4

Sequential. Computing, for each $q \in \{1, \dots, m_q\}$, the factors $T_{q,j} = (1 - \tau_q) + \tau_q u_j$ and the associated quantities $\log |T_{q,j}|$ and $\operatorname{sgn}(T_{q,j})$ costs $O(d\, m_q)$ arithmetic operations in total. Accumulating $\log |P_q| = \sum_j \log |T_{q,j}|$ and $\operatorname{sgn}(P_q) = \prod_j \operatorname{sgn}(T_{q,j})$ is a length-d reduction for each of the m_q nodes, contributing $O(d\, m_q)$ additional work. Evaluating equation (8) then requires, for each $i \in \{1, \dots, d\}$, a weighted sum of m_q terms, each obtained from $\log |P_q| - \log |T_{q,i}|$ in constant time, for a further $O(d\, m_q)$ operations. The total is therefore $O(d\, m_q)$.

Parallel time. Given an associative operator $\oplus$ and a length-d sequence, a tree reduction, the reduction case of the associative scan, evaluates $a_1 \oplus \cdots \oplus a_d$ in parallel time $O(\log d)$ using $O(d)$ processors (Blelloch, 1990). Both $\log |P_q|$ (under addition) and $\operatorname{sgn}(P_q)$ (under multiplication) are such reductions over the feature axis of length d, and the m_q nodes are independent, so the shared-product phase completes in parallel time $O(\log d)$. For each i, evaluating equation (8) requires (i) subtracting $\log |T_{q,i}|$ from $\log |P_q|$ pointwise (parallel time $O(1)$), and (ii) a weighted sum over q, which is a length-m_q tree reduction with parallel time $O(\log m_q)$. Since the d Shapley values are independent given $\{(\log |P_q|, \operatorname{sgn}(P_q))\}_{q=1}^{m_q}$, they are evaluated in parallel across i without increasing parallel time. The overall parallel time is therefore $O(\log d + \log m_q) = O(\log d)$, as $m_q \le \lceil d/2 \rceil$. The required number of processors is the ratio of total work to parallel time (Brent, 1974), giving $O\big(d\, m_q / (\log d + \log m_q)\big)$ processors. The exact regime $m_q = \lceil d/2 \rceil$ is a special case. □

PROOF OF PROPOSITION 5

The empirical interventional value function replaces the expectation in $v_{\mathbf{x}}^{\mathrm{int}}(\mathcal{S}) = \mathbb{E}_{X_{\mathcal{D}\setminus\mathcal{S}}}\big[f(\mathbf{x}_{\mathcal{S}}, X_{\mathcal{D}\setminus\mathcal{S}})\big]$ by the average over the background dataset $\mathcal{B}$,

$$\hat{v}_{\mathbf{x}}(\mathcal{S}) = \frac{1}{n_b} \sum_{b=1}^{n_b} f\big(\mathbf{x}_{\mathcal{S}}, \tilde{\mathbf{x}}_{\mathcal{D}\setminus\mathcal{S}}^{(b)}\big),$$

where $(\mathbf{x}_{\mathcal{S}}, \tilde{\mathbf{x}}_{\mathcal{D}\setminus\mathcal{S}}^{(b)})$ denotes the point whose j-th coordinate is x_j for $j \in \mathcal{S}$ and $\tilde{x}_j^{(b)}$ for $j \notin \mathcal{S}$. Fix b and $\mathcal{S}$ and substitute equation (9). Each factor $u_j^{(r)}$ depends on the j-th coordinate only, so at the mixed point it evaluates to $u_j^{(r)}(x_j) = u_j^{(r)}$ if $j \in \mathcal{S}$ and to $u_j^{(r)}(\tilde{x}_j^{(b)}) = \tilde{u}_j^{(rb)}$ if $j \notin \mathcal{S}$. Hence

$$f\big(\mathbf{x}_{\mathcal{S}}, \tilde{\mathbf{x}}_{\mathcal{D}\setminus\mathcal{S}}^{(b)}\big) = \sum_{r=1}^{p} c_r \prod_{j \in \mathcal{D}} u_j^{(r)}\big((\mathbf{x}_{\mathcal{S}}, \tilde{\mathbf{x}}_{\mathcal{D}\setminus\mathcal{S}}^{(b)})_j\big) = \sum_{r=1}^{p} c_r \prod_{j \in \mathcal{S}} u_j^{(r)} \prod_{j \notin \mathcal{S}} \tilde{u}_j^{(rb)}.$$

Averaging over b and exchanging the two finite sums gives equation (10). For each pair (r, b) the summand is a product with one factor per feature, $u_j^{(r)}$ if $j \in \mathcal{S}$ and $\tilde{u}_j^{(rb)}$ if $j \notin \mathcal{S}$, and the weights c_r / n_b and the absent factors $\tilde{u}_j^{(rb)}$ do not depend on $\mathcal{S}$; thus $\hat{v}_{\mathbf{x}}$ is a weighted sum of $p\, n_b$ such games. When every absent factor is non-zero, dividing the (r, b)-th summand by $\prod_{j \in \mathcal{D}} \tilde{u}_j^{(rb)}$ exhibits it as the product game of Definition 1 with factors $u_j^{(r)} / \tilde{u}_j^{(rb)}$; the general case, in which some $\tilde{u}_j^{(rb)}$ may vanish, is handled without division by Proposition 6. □

PROOF OF PROPOSITION 6

By linearity of the Shapley value it suffices to treat one component–background pair, that is the game $w(\mathcal{S}) = \prod_{j \in \mathcal{S}} u_j \prod_{j \notin \mathcal{S}} \tilde{u}_j$ for arbitrary $u_j, \tilde{u}_j \in \mathbb{R}$, where the superscripts (r) and (rb) are omitted.

Its marginal contributions are

$$w(\mathcal{S}\cup\{i\})-w(\mathcal{S})=(u_i-\tilde{u}_i)\prod_{j\in\mathcal{S}}u_j\prod_{j\notin\mathcal{S}\cup\{i\}}\tilde{u}_j.$$

Substituting the Beta representation $\mu(s)=\int_0^1 t^s(1-t)^{d-s-1}\,dt$ of the Shapley weights into equation (1) and exchanging the finite sum with the integral,

$$\phi_i(w)=(u_i-\tilde{u}_i)\int_0^1\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{i\}}\prod_{j\in\mathcal{S}}t\,u_j\prod_{j\in\mathcal{D}\setminus\mathcal{S}\setminus\{i\}}(1-t)\,\tilde{u}_j\,dt=(u_i-\tilde{u}_i)\int_0^1\prod_{j\neq i}\big((1-t)\tilde{u}_j+t\,u_j\big)\,dt,$$

since the subset sum is precisely the expansion of the product over the $d-1$ features $j\neq i$. No factor is inverted, so the identity holds also when some $\tilde{u}_j$ vanish. The integrand is a polynomial of degree at most $d-1$ in t, so the m_q-point Gauss–Legendre rule evaluates the integral exactly as soon as $2m_q-1\geq d-1$, that is $m_q\geq\lceil d/2\rceil$ (Davis & Rabinowitz, 1984). Summing over the $p\,n_b$ pairs with the weights c_r/n_b gives equation (11). □

F TREE-BASED MODELS

This appendix collects the material on tree ensembles: the path-dependent value function and its leaf-wise product structure (Section F.1), and the sequential evaluation and its relation to the other quadrature-based tree explainers (Section F.2). The CPU and GPU benchmarks are in Section G.5.

F.1 PATH-DEPENDENT VALUE FUNCTION

Consider a decision tree with leaf set $\mathcal{L}$ and edge set $\mathcal{E}$. Each leaf $\ell\in\mathcal{L}$ stores a prediction ν_ℓ, and $\mathcal{E}_\ell\subseteq\mathcal{E}$ denotes the edges on the root-to-ℓ path. In the path-dependent (cover-based) value function of TreeSHAP (Lundberg et al., 2020), a coalition $\mathcal{S}$ is treated as observed by resolving the splits on features in $\mathcal{S}$ deterministically for $\mathbf{x}$, while splits on features outside $\mathcal{S}$ are averaged with the branch probabilities. For an edge $e\in\mathcal{E}$ let $p_e\in(0,1)$ be the fraction of training samples taking that edge and $\kappa(e)\in\mathcal{D}$ the feature of the split, and let $c_e(\mathbf{x})=1/p_e$ if $\mathbf{x}$ follows e and $c_e(\mathbf{x})=0$ otherwise. Grouping the edges of a path by feature defines the path factor

$$q_{j,\ell}(\mathbf{x})=\prod_{e\in\mathcal{E}_\ell:\,\kappa(e)=j}c_e(\mathbf{x}),$$

with the empty product equal to one, so that $q_{j,\ell}(\mathbf{x})=1$ for every feature not tested on the path. The value function is then

$$v_{\mathbf{x}}(\mathcal{S})=\sum_{\ell\in\mathcal{L}}\nu_\ell\Big(\prod_{e\in\mathcal{E}_\ell}p_e\Big)\prod_{j\in\mathcal{S}}q_{j,\ell}(\mathbf{x}),\tag{33}$$

a weighted sum of product games in the sense of Definition 1, one per leaf, with factors $u_j=q_{j,\ell}(\mathbf{x})$ and weights $\nu_\ell\prod_{e\in\mathcal{E}_\ell}p_e$; since the branch probabilities are positive, the normalization of Definition 1 is always available here. Only the features tested on the path to ℓ enter the product, so the degree of the leaf's integrand is the number η_ℓ of distinct features on that path, at most the path length and unchanged by repeated splits on the same feature; we write $\eta=\max_\ell\eta_\ell$. Ensembles follow by linearity.

Applying Proposition 2 leaf by leaf, with $a_r(q)=1-\tau_r+\tau_r q$, $s_r(q)=(q-1)/a_r(q)$ and $G_\ell(r)=\nu_\ell\big(\prod_{e\in\mathcal{E}_\ell}p_e\big)\prod_{j\in\mathcal{D}}a_r\big(q_{j,\ell}(\mathbf{x})\big)$,

$$\widehat{\phi}_i=\sum_{r=1}^{m_q}\omega_r\sum_{\ell\in\mathcal{L}}G_\ell(r)\,s_r\big(q_{i,\ell}(\mathbf{x})\big),\tag{34}$$

which is exact for $m_q\geq\lceil\eta/2\rceil$ irrespective of the number of features d. The factor s_r divides the contribution of feature i out of the shared product $G_\ell(r)$, exactly as in Section 3.3; the division is safe because $a_r(q)\geq 1-\tau_r>0$ for $q\geq 0$.

F.2 Sequential evaluation and relation to other quadrature-based tree explainers

Evaluating equation (34) leaf by leaf costs $O(m_q|\mathcal{L}|\eta)$, or $O(|\mathcal{L}|\eta^2)$ in the exact regime, the cost of the original TreeSHAP. The redundancy across leaves is removed by the telescoping reorganization of Linear TreeSHAP (Yu et al., 2022a), which carries over to the quadrature representation unchanged. Traversing an edge e with $\kappa(e) = i$ changes the state of feature i from q_e^- to $q_e^+ = q_e^-\, c_e(\mathbf{x})$, and the leaf-wise sum in equation (34) can be reindexed over edges: with $H_e(r) = \sum_{\ell \text{ below } e} G_\ell(r)$ and $\Delta_e(r) = s_r(q_e^+) - s_r(q_e^-)$,

$$\widehat{\phi}_i = \sum_{r=1}^{m_q} \omega_r \sum_{e \in \mathcal{E}:\, \kappa(e)=i} H_e(r)\, \Delta_e(r). \tag{35}$$

A single depth-first traversal maintains the current path state, aggregates the leaf weights $H_e(r)$ upward and applies the increment $\Delta_e(r)$ once per edge, so every edge costs $O(m_q)$ and the whole tree $O(m_q|\mathcal{L}|)$, that is $O(|\mathcal{L}|\eta)$ in the exact regime; the leaf–feature table $q_{i,\ell}$ is never materialized. This is the complexity of Linear TreeSHAP without its interpolation step, which is what removes the instability documented below.

On a CPU this computation is the same for all three quadrature-based tree explainers. Our tree instantiation was developed independently of TreeGrad-Shap (Li et al., 2026), which appeared while this work was under development, and Quadrature-TreeSHAP (Wettenstein et al., 2026) is concurrent; all three evaluate the same leaf-wise polynomial at the same Gauss–Legendre nodes (Section A), the telescoping simplification applies to each, and they differ only in the node budget (Quadrature-TreeSHAP fixes eight nodes, the other two use the exact count) and in implementation. The stability that quadrature brings over interpolation-based coefficient recovery is therefore shared by all three, and a CPU comparison among them would compare implementations of one algorithm; the CPU benchmark below is accordingly against the established tree explainers. On the GPU the implementations differ substantially. TreeGrad-Shap has no GPU implementation, and we compare our native CUDA implementation with the CUDA backend of Quadrature-TreeSHAP in XGBoost and with GPUTreeSHAP (Mitchell et al., 2022b); since the first two evaluate the same quadrature representation, the differences between them measure the CUDA implementations rather than the method.

G Experimental Details and Additional Results

This appendix collects the experimental material behind Section 5: the conventions shared by all benchmarks (Section G.1), the datasets and fitted models (Section G.2), the certified-budget study of Section 5.1 (Section G.3), the additional comparison of product-kernel explainers under the interventional value function (Section G.4), and the CPU and GPU benchmarks of the tree instantiation with their full tables (Section G.5). The experiments were run on one workstation with an Intel Core Ultra 9 285K processor, 192 GB of system memory, and a single NVIDIA GeForce RTX 5090 GPU with 32 GB of memory; all GPU arithmetic on this machine is float64 unless otherwise stated.

G.1 Common protocol

Within each benchmark every method explains the same value function on the same fitted model and the same inputs, so the methods' outputs are estimates of one Shapley vector and differ only in accuracy and cost. Reported times are end-to-end wall-clock times of the public explanation call; model training, model conversion and explainer construction are excluded unless stated otherwise, and the statistic (median or mean over inputs, seeds or repetitions) is stated with each table. Accuracy is measured against a reference on the same game: the maximum absolute coordinate difference from independent or high-node-count reference Shapley values where such a reference is computed (Sections G.3 and G.4), and the worst-case violation of the efficiency axiom, which any exact implementation satisfies to rounding, where the models are too large for an external reference (Section G.5). For QuadraSHAP, the *certified* node count is the one selected from the a priori bound of Section C before any computation, and the *observed* count is the smallest tested count

whose measured error meets the same tolerance; references enter only the latter. Arithmetic is float64 unless stated.

G.2 Datasets and models

G.2.1 Synthetic data

Two synthetic generators are used, one for the node-budget study and one for the product-kernel benchmark.

Node-budget study (Section 5.1, Section G.3). For each $d \in \{50, 100, 250, 500, 1000\}$ and each seed $s \in \{0, \dots, 4\}$, the features are independent standard normal, $X_{ij} \sim \mathcal{N}(0, 1)$, and a coefficient vector $\beta \in \mathbb{R}^d$ has exactly $0.3d$ nonzero entries at random positions with $\beta_j = \pm 0.2$ and random signs; the remaining 70% of the coordinates carry no signal. With $\eta = X\beta$, the response depends on the model family: $y = \eta/\|\beta\|_2 + 0.2\,\xi$ with $\xi \sim \mathcal{N}(0, 1)$ for the regression explained by KRR; counts $y \sim \text{Poisson}(e^{\eta})$; binary labels $y \sim \text{Bernoulli}(\sigma(\eta))$ with σ the logistic function; and, for Naive Bayes, balanced labels $y \sim \text{Bernoulli}(1/2)$ with class-conditional means $\pm\beta/2$ and unit variances. Each fit uses $4{,}000$ training points and a held-out block of 500 points, of which the first 20 are explained. The fixed-total-signal control of Section G.3.2 uses $\beta_j = \pm 0.2\sqrt{100/d}$ instead.

Product-kernel benchmark (Section 5.2). For each $d \in \{50, 500, 1000, 2000, 5000\}$ we draw $n = 1{,}000$ samples with `sklearn.datasets.make_regression` (n_informative $= \lfloor d/4 \rfloor$, noise $= 0.1$, random_state $= 42$), split them 80/20 (shuffled, seed 42), and standardize features and target on the training part. The 50 explained inputs per d are fixed held-out points.

G.2.2 Models on synthetic data

Table 6 lists the models of the node-budget study. All are `scikit-learn` estimators with the stated hyperparameters and no tuning, and the quantity explained is the output on which each model is multiplicative (Table 1). Absent features are replaced by the baseline $\mathbf{x}^b = 0$, the population mean of every design, so the four families are explained on the same kind of baseline game; the interventional control of Section G.3.2 uses the first eight training rows as background instead. The product-kernel benchmark fits an RBF kernel ridge regressor with $\gamma = 1/d$ and ridge parameter 0.1 on the standardized training part of each dataset and explains the neutral-factor game (Section D).

Table 6: Models of the node-budget study: `scikit-learn` estimator with its non-default hyperparameters, the quantity explained, and the training-set size.

Family	Estimator	Quantity explained	n_{train}
KRR	`KernelRidge(kernel="rbf", gamma=1/d, alpha=0.1)`	prediction	4,000
Poisson regression	`PoissonRegressor(alpha=0.1, solver="newton-cholesky")`	mean	4,000
Logistic regression	`LogisticRegression(C=0.1)`	odds	4,000
Gaussian Naive Bayes	`GaussianNB()`	posterior odds	4,000

G.2.3 Text-classification datasets and models

All five datasets used in the real-world tree ensemble benchmark (Section G.5) and the product-kernel benchmark (Section 5.2) are loaded directly from the Hugging Face `datasets` library. Table 7 summarizes their basic statistics.

Table 7: Statistics of the five text-classification datasets.

Dataset	Train	Test	Classes	Task
`emotion`	16,000	2,000	6	Emotion recognition
`imdb`	25,000	25,000	2	Sentiment analysis
`sms_spam`	4,459	1,115	2	Spam detection
`sst2`	67,349	872	2	Sentiment analysis
`rotten_tomatoes`	8,530	1,066	2	Sentiment analysis

Text preprocessing. Documents are tokenized with TF-IDF using the following fixed configuration: lower-casing, English stop-word removal, unigram and bigram features (ngram_range=$(1, 2)$), minimum document frequency min_df=3 (min_df=2 for `sms_spam`, whose 5,574 messages otherwise yield fewer than 5,000 terms), maximum document frequency max_df=0.95, sublinear TF scaling (sublinear_tf=True), and vocabulary capped at 5,000 terms (max_features=5000). This yields a d=5,000 dimensional TF-IDF feature vector for each document.

Tree ensemble models. For the tree ensemble benchmark, a `RandomForestClassifier` is trained on each dataset with `n_estimators = 100` (`emotion`, `imdb`, `sms_spam`) or `300` (`sst2`, `rotten_tomatoes`), `max_depth = 100`, `min_samples_leaf = 1`, and `random_state = 42`. No hyperparameter search is performed for the tree models; the configuration is fixed across all datasets to ensure a fair stability comparison at large depths.

SVM model. For the product-kernel benchmark (Section 5.2) and the matched interventional comparison (Section G.4), one support vector classifier with RBF kernel is trained per dataset on the full training split with fixed hyperparameters C=2 and γ="scale"; no per-dataset tuning is performed, so that every explained text of a corpus is scored by the same model. For `emotion` the explained score is the one-vs-rest decision value of the predicted class. Held-out accuracies are 88.0% (`imdb`), 76.0% (`rotten_tomatoes`), 78.1% (`sst2`, validation split), 98.2% (`sms_spam`) and 89.8% (`emotion`); they describe the models, not the explainers. Twenty fixed held-out texts per corpus are explained, complete and untruncated (`imdb` test reviews have 118–582 words, median 216).

G.2.4 Survival cohorts and Cox models

Cohorts. GSE24080 (MAQC Consortium, 2010) contains Affymetrix expression profiles of multiple-myeloma patients. We use the deposited MAS5 $\log_2$ intensities of all 54,675 probe sets without gene-level aggregation, the continuous overall-survival (OS) times in months, and the provider's training/validation split; after removing the five provider-flagged outlier profiles and one patient with zero follow-up, 553 patients remain (339 training, 214 validation, 168 deaths). TCGA lower-grade glioma (LGG) (Cancer Genome Atlas Research Network, 2015) contributes Illumina HumanMethylation450 beta values from UCSC Xena with OS outcomes from the TCGA Clinical Data Resource. Primary tumors with a valid positive OS time give 511 patients (125 deaths), and removing the 89,512 sites that are missing for every patient leaves $d = 396{,}065$ sites; the cohort is split 75/25 stratified by event status (seed 42) into 383 training and 128 test patients. Preprocessing is fitted on the training patients only: remaining missing values are imputed by training medians, and features are centered and scaled with training statistics. No feature is constant on the training set, so all d features are retained.

Cox models. Each cohort is modeled by a Cox proportional-hazards model $h(t \mid \mathbf{x}) = h_0(t)\exp(\beta^\top \mathbf{x})$ with Breslow handling of tied event times, fitted with `scikit-survival` under a ridge penalty fixed in advance at $0.01d$ (546.75 and 3,960.65) without tuning. Because $d \gg n$, the fit is carried out in the span of the training rows, a change of coordinates rather than a feature selection, so all d coefficients are nonzero. Test concordance indices are 0.66 (GSE24080) and 0.85 (TCGA LGG); they describe the models, not the explanations. The quantity explained is the relative hazard $\exp(\hat{\beta}^\top \mathbf{x}) = \prod_j \exp(\hat{\beta}_j x_j)$, a single product game ($p = 1$) with factors $u_j = \exp(\hat{\beta}_j x_j)$ (Table 1).

Explained patients and background. The empirical interventional value function (Proposition 5) uses $n_b = 30$ training patients as background, the first 30 entries of a seed-42 permutation of the training rows, uniformly weighted and held fixed across all node counts. The explained inputs are the first five held-out patients of each split, chosen without regard to outcome. The exactness threshold $\lceil d/2 \rceil$ is 27,338 nodes for GSE24080 and 198,033 for TCGA LGG.

G.3 Synthetic experiments on certified quadrature budgets

This section gives the protocol and additional tests for Section 5.1. The central question is practical: *how many quadrature nodes are enough to compute every Shapley value of an input to a specified accuracy?* A model with d features produces d Shapley values, but the number of integration nodes

m_q need not equal $\lceil d/2 \rceil$ to get a highly accurate estimation. We compare the node count selected by the mathematical error bound (*certified*) with the smallest tested count that meets the same tolerance in the experiment (*observed*).

The primary target is maximum absolute attribution error 10^{-6}. Section G.3.1 explains the measurement and its numerical validation; Section G.3.2 tests why the required count changes; and Section G.3.3 isolates the effect of factor strength in a game whose Shapley values are known analytically.

G.3.1 Measuring observed and certified node counts

What is held fixed, and what changes? For each model family and feature count $d \in \{50, 100, 250, 500, 1000\}$, we fit a model and then explain held-out inputs. We keep that fitted model and each input fixed while changing m_q. A new feature count requires a new fit, but a new node count does not. Exactly $k = 0.30d$ coordinates are informative in the data-generating model (Section G.2.1; models in Section G.2.2); the fitted models are not constrained to have only k nonzero coefficients.

Two node counts for the same input. Let $\widehat{\phi}_i^{(m_q)}(x)$ be the Shapley approximation for feature i using m_q nodes, and let $\phi_i^{\mathrm{ref}}(x)$ be the exact computation. The measured error is

$$E_m(x) = \max_{1 \le i \le d} \left| \widehat{\phi}_i^{(m_q)}(x) - \phi_i^{\mathrm{ref}}(x) \right|. \tag{36}$$

For the *certified* count, Algorithm 2 computes the bound $C_m(x) = A_{\max}(x) B(m, \bar{\Lambda}(x))$ from the fitted model's factor tables and selects the smallest integer count for which it is at most ϵ. If no smaller count suffices, it returns $\lceil d/2 \rceil$, where the polynomial quadrature is exact. This selection does not use ϕ^{ref} or the observed errors.

For the *observed* count, we evaluate the approximation with successively larger integer counts and compare it with the reference:

$$m_{\mathrm{obs}}(x) = \min\{m \in \mathcal{M}(x) : E_m(x) \le \epsilon\}, \tag{37}$$

where $\mathcal{M}(x)$ is the tested consecutive-node range. This tells us how many nodes were sufficient for that input in the measured experiment. Computing it requires the exact SV; it is not a reference-free stopping rule for new inputs. In use, the certified count can be selected without running this comparison.

Both counts concern the same model, input, absent-feature reference, and tolerance. In particular, neither curve is a competing explanation method. For example, the logistic panel of Figure 1 has median counts of 10 observed and 43 certified at $d = 1000$: the measured errors meet the target with substantially fewer nodes than the bound requires. The certificate is conservative, but both counts are small compared with 500-node exactness. The median is taken separately for each quantity; these are not necessarily the counts of one particular input.

Repetitions and plotted summaries. Each configuration uses five independent data/training seeds (0–4). Each fit has a 500-point held-out test block; its first 20 inputs are explained without filtering. Each plotted model–dimension setting therefore has $5 \times 20 = 100$ explanations, grouped within five training repetitions. Curves show medians; shaded bands and error bars show the 10th–90th percentiles across these inputs. These describe variation among explanations, not confidence intervals for an average. No hyperparameter selection uses the held-out test block.

Sweep and reference calculations. For each input, we test every integer from one through $\min\{\lceil d/2 \rceil, m_{\mathrm{cert}}(10^{-9}) + 5\}$, as well as the exactness-threshold count. We retain all results and evaluate tolerances 10^{-3}, 10^{-5}, 10^{-6}, and 10^{-9}. The certificate uses the full-coordinate variation $\bar{\Lambda}$ in equation (24); neither $\bar{\Lambda}$ nor $A_{\max}$ is fixed or calibrated to produce a desired node count.

Results across inputs. The main figure is in Section 5.1; Table 8 adds the spread and the largest certified counts at $d = 1000$. At $d = 50$ the median certified count is five for all four families, and no input needs more than six. At $d = 1000$ the median certified counts are 24 (Poisson), 43 (logistic),

and 30 (NB), with maxima of 26, 47, and 34 over the 100 inputs of each family; KRR stays at a median of four and a maximum of four. In hindsight, no input at $d = 1000$ needs more than 14 nodes to reach $\epsilon = 10^{-6}$, so per input the certificate over-provisions by a factor between 1.3 and 5.1. KRR is the case where increasing d does *not* increase the node count: with $\gamma = 1/d$ its total variation stays near $\Lambda \approx 2$ at every d, whereas Λ grows to 45–91 for the other three families.

Table 8: Node counts at $d = 1000$, $\epsilon = 10^{-6}$. Each row summarizes 100 explanations from five fits; the largest observed count is the smallest m_q that met the tolerance for every input, known only in hindsight. The full-dimensional exactness threshold is 500.

Model	Largest observed	Certified median	Certified 10th–90th	Largest certified
KRR	3	4	4–4	4
Poisson	10	24	22–24	26
Logistic	14	43	41–45	47
Gaussian NB	11	30	28–32	34

Variation across fits. To check that the growth with d is not an artifact of a single fit, we take, for each family and seed, the median certified count at $d = 50$ and at $d = 1000$ and record the increase. Averaged over the five seeds, the increase is 19.2 nodes for Poisson (95% bootstrap interval $[19.0, 19.4]$), 37.7 for logistic ($[36.7, 38.9]$), and 24.8 for NB ($[24.6, 25.0]$), the intervals obtained by resampling the five seeds 10,000 times. The growth is thus consistent across fits.

Error curves and numerical precision. The node-budget figure fixes an error tolerance and asks how many quadrature nodes are sufficient. Figure 2 asks the complementary question: for a given node count, how large are the observed error and its certified bound? We show the results for $d = 1000$. The observed errors (blue) fall below the target 10^{-6} before the certified bounds (orange), illustrating the conservativeness of the certified node-selection rule. As m_q increases, the quadrature truncation error initially dominates and the observed error decreases rapidly. Eventually, the truncation error becomes smaller than the numerical error introduced by float64 products and summations. Additional nodes then provide little practical improvement, and the observed error approaches a numerical plateau. The gray region indicates the approximate numerical-resolution scale identified by float64 consistency and efficiency checks; it is a diagnostic, not a rigorous floating-point error bound. The plateau is model dependent: RBF factors remain in a numerically mild range, producing errors near 10^{-16}, whereas the larger intermediate products and cancellation arising in Poisson means, logistic odds, and Naive Bayes odds lead to plateaus around 10^{-14}–10^{-12}. The certified bound can continue decreasing below these plateaus because it controls quadrature truncation in exact arithmetic, not finite-precision evaluation.

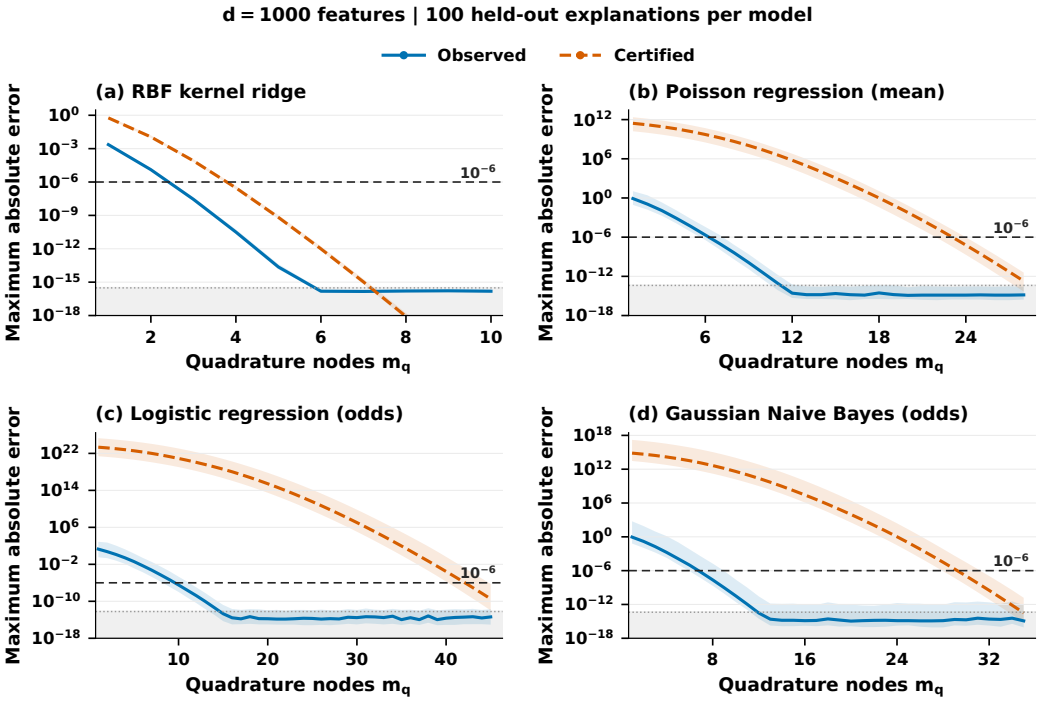

Figure 2: **Error versus node count at $d = 1000$.** The horizontal axis here is m_q and the vertical axis is error. Blue: maximum absolute attribution error against the exact vector. Orange: the certified bound. Curves are medians and bands are 10th–90th percentiles over 100 explanations per family. The dashed horizontal line is the target 10^{-6}. Panels have different vertical scales and show the consecutive-node range available for every input in that panel. Gray shading marks a median reference-discrepancy diagnostic, not a rigorous floating-point error bound.

G.3.2 Separating feature count, signal strength, and reference choice

Control 1: separating feature count from signal strength. In the main experiment, increasing the number of features also makes the underlying prediction problem stronger. At every dimension, 30% of the features are informative, and every informative feature has a generating coefficient of magnitude 0.20. Consequently, a model with more features also has more informative features of the same strength. Because the features are independent and standardized, the total signal in the linear predictor is measured by

$$\|\beta\|_2^2 = (0.30d)(0.20)^2 = 0.012d.$$

Thus, the total signal increases from 1.2 at $d = 100$ to 12 at $d = 1000$. We call this the *fixed per-feature signal* condition: each informative feature remains equally strong, so adding features increases both dimensionality and total signal.

To separate these two effects, we repeat the experiment for the three non-RBF model families while keeping the total signal constant. Specifically, we reduce the magnitude of each informative coefficient to

$$0.20\sqrt{100/d}.$$

There are still $0.30d$ informative features, but each feature becomes weaker as d grows, giving

$$\|\beta\|_2^2 = (0.30d)\left(0.20\sqrt{100/d}\right)^2 = 1.2$$

for every dimension. We call this the *fixed total signal* condition. The two conditions are identical at $d = 100$, where both use coefficient magnitude 0.20; therefore, the repeated results at this dimension do not constitute additional independent evidence. All other data-generation, training, and explanation settings are unchanged.

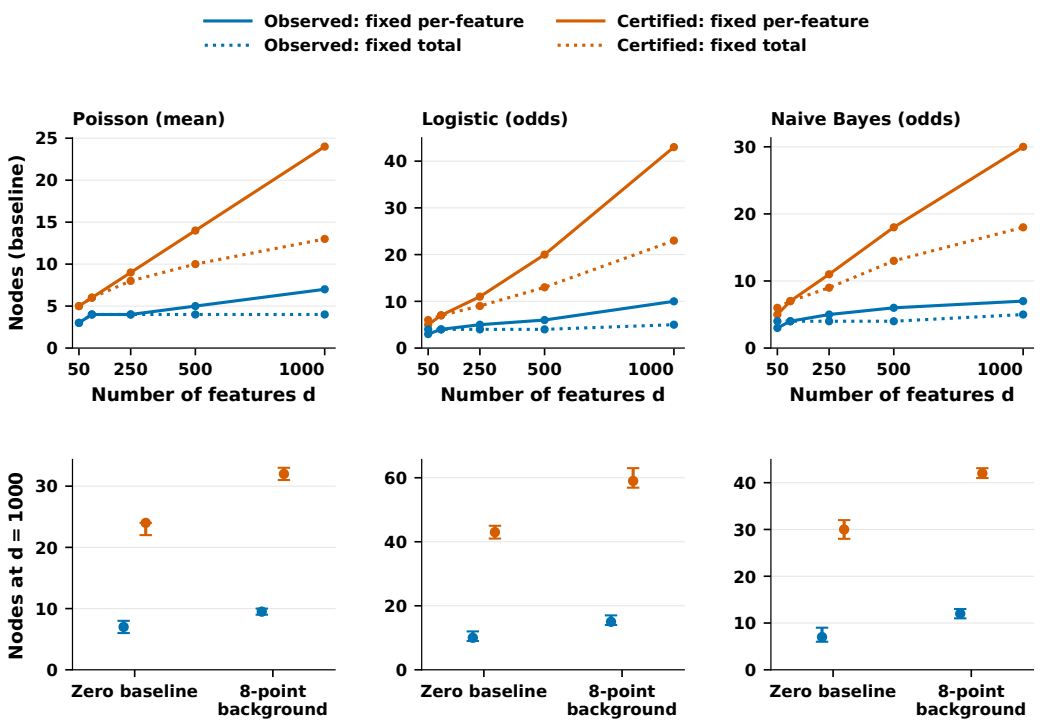

Figure 3: **Two controls for the required node count at $\epsilon = 10^{-6}$.** Columns are Poisson mean, logistic odds, and Gaussian NB odds. *Top:* median counts as d increases, with either fixed per-feature signal (solid) or fixed total signal (dotted); both use a zero baseline. *Bottom:* counts at $d = 1000$ for the same main-design models and inputs, comparing zero replacement with an eight-point training background. Blue denotes observed counts and orange certified counts in both rows. Each setting has five fits and 20 inputs per fit; bottom-row error bars are 10th–90th percentiles, not confidence intervals.

The top row of Figure 3 compares the two conditions. Solid lines represent fixed per-feature signal, while dotted lines represent fixed total signal. Blue shows the observed node count required to attain error below 10^{-6}, and orange shows the certified node count. At $d = 1000$, the observed/certified median counts decrease from $7/24$ to $4/13$ for Poisson regression, from $10/43$ to $5/23$ for logistic regression, and from $7/30$ to $5/18$ for Gaussian Naive Bayes when total signal is held fixed. Thus, the growth in node count in the main experiment is not caused by feature count alone: part of it arises because adding equally strong informative features also increases the total signal and makes the resulting product game more numerically demanding. The control does not completely isolate dimension, however, because the models are refitted at every dimension, so estimation noise, fitted coefficients, and the parameters entering the certificate may still change.

Control 2: baseline versus interventional baseline. We next examine how the choice of value function affects the required number of quadrature nodes. Under the *baseline* value function, every absent feature is replaced by zero. Under the *interventional baseline*, absent features are instead replaced using eight training samples. For each of these eight samples, we evaluate the model after replacing the absent features by that sample's feature values, and then average the eight model outputs. We perform this comparison at $d = 1000$ using exactly the same fitted models and test inputs. Thus, only the value function changes; the predictive model and the input being explained remain fixed. Because the two value functions define different coalition games, they can produce different Shapley values and require different quadrature budgets.

The lower row of Figure 3 presents the results for the fixed per-feature signal condition. Moving from the baseline to the interventional baseline increases the median observed node count from 7 to 9.5 for Poisson regression, from 10 to 15 for logistic regression, and from 7 to 12 for Gaussian Naïve Bayes.

The corresponding median certified counts increase from 24 to 32, from 43 to 59, and from 30 to 42, respectively. The largest certified counts under the interventional baseline are 34, 64, and 46. Although the interventional value function is more demanding, these budgets remain much smaller than the exactness threshold $\lceil d/2 \rceil = 500$. For example, logistic regression requires a median of 59 certified nodes, only 11.8% of the exactness threshold. The value 9.5 is possible because it is the median of an even number of integer-valued node counts.

These certificates apply to the finite interventional game defined by the eight selected training samples. They bound its quadrature error, but they do not bound the statistical error incurred by using eight samples to represent the full background population.

G.3.3 A CONTROLLED PRODUCT GAME WITH KNOWN SHAPLEY VALUES

The preceding experiments use fitted models, so several quantities change simultaneously as the number of features increases, including the fitted coefficients, prediction scale, and product factors. We therefore construct a deterministic product game that isolates the effect of accumulating factor changes. Its Shapley values are available in closed form, so no model training or numerical reference calculation is required.

Let J contain the $k = 0.30d$ active features, while the remaining features have no effect. For a factor ratio $r \in \{1.1, 1.5, 3, 10\}$, define

$$v(S) = r^{|S \cap J| - k}, \qquad \phi_i = \begin{cases} (1 - r^{-k})/k, & i \in J, \\ 0, & i \notin J. \end{cases} \tag{38}$$

When no active feature is present, the coalition value is r^{-k}. Adding an active feature multiplies the value by r, and including all active features gives $v(\mathcal{D}) = 1$. The parameter r therefore controls how strongly the coalition value changes when a feature is added. Because all active features play identical roles, they divide the total change

$$v(\mathcal{D}) - v(\varnothing) = 1 - r^{-k}$$

equally, giving the Shapley values in equation (38). Inactive features have zero Shapley value.

All coalition values lie in $[0, 1]$, regardless of d and r. Consequently, any increase in the required node count cannot be attributed to an exploding output scale. For numerical stability, we implement each active feature using a present factor of 1 and an absent factor of $1/r$, which produces exactly the game in equation (38) while avoiding unnecessary underflow.

We evaluate the game at $d \in \{50, 100, 250, 500, 1000\}$ and target error 10^{-6}. For each setting, we record the smallest node count that attains the target using the analytic Shapley values, together with the node count selected by the certificate.

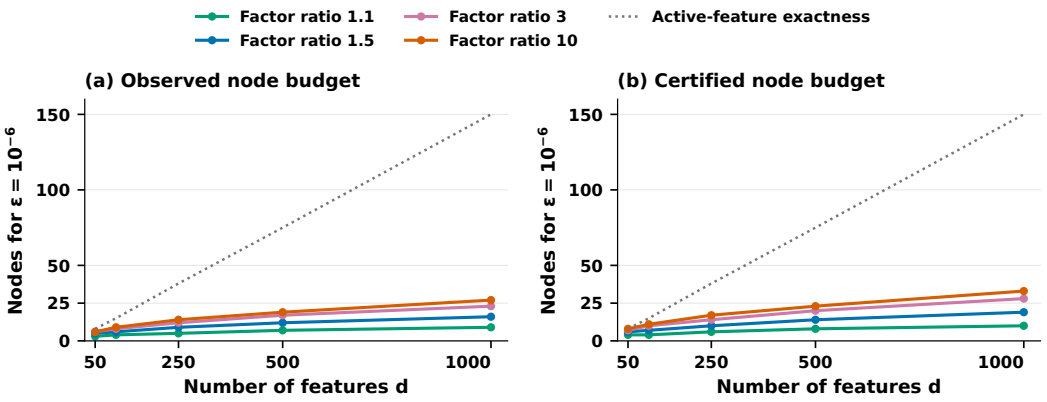

Figure 4: **Node counts for the controlled product game.** The left panel reports the observed node count required to attain error below 10^{-6}, using the analytic Shapley values in equation (38). The right panel reports the corresponding certified count. The horizontal axis is the number of features d, and colors identify the factor ratio r. Only $k = 0.30d$ features are active, so the relevant exactness threshold is $\lceil k/2 \rceil$, shown in gray, rather than $\lceil d/2 \rceil$. The five dimensions and four factor ratios give 20 deterministic cases; there are no training repetitions or error bars.

As either d or r increases, more active factors change in the same direction and the game requires more quadrature nodes. Formally, the amplitude term remains bounded,

$$A_{\max} = 1 - \frac{1}{r},$$

while the cumulative factor variation,

$$\bar{\Lambda} = k\left(1 - \frac{1}{r}\right),$$

increases with the number of active features. Thus, the experiment increases the cumulative difficulty of the product game without increasing its output range.

For the strongest factor change, $r = 10$, the observed count rises from 6 at $d = 50$ to 27 at $d = 1000$, while the certified count rises from 8 to 33. At $d = 1000$, the observed counts for $r = 1.1$, 1.5, 3, and 10 are 9, 16, 23, and 27, and the corresponding certified counts are 10, 19, 28, and 33.

At $d = 1000$, only $k = 300$ features are active, so the exactness threshold is $\lceil k/2 \rceil = 150$. The largest certified budget, 33, is therefore only 22% of the exactness threshold. At the smallest setting, $d = 50$ and $r = 10$, the certified count equals the active-feature threshold of 8, so the method does not provide a strict node reduction in every case. Overall, the experiment shows that the required budget can grow as more factors contribute, while still remaining much smaller than the exact quadrature budget at high dimension. This controlled result does not establish the same behavior for arbitrary signed factors, different output scalings, dimensions above 1000, or every model and value function.

G.4 Interventional explanations for product-kernel models

We extend the text-classification experiment in Section 5.2 to the interventional value function, comparing QuadraSHAP with KernelSHAP and SamplingSHAP (Lundberg & Lee, 2017). We reuse the fitted product-kernel models and held-out texts from the main experiment. All three methods explain the same model output using the same background set, making their attribution errors directly comparable within this experiment.

Experimental setup. We use the 5,000-feature RBF SVCs for RT, `sst2`, `sms spam`, and `emotion`. Each corpus contributes 20 held-out texts and one fixed, class-stratified background of 30 training texts (15 per class for the binary tasks and five per class for `emotion`), giving 80 explained texts overall. IMDB is left out because its long reviews require the sampling estimators to draw

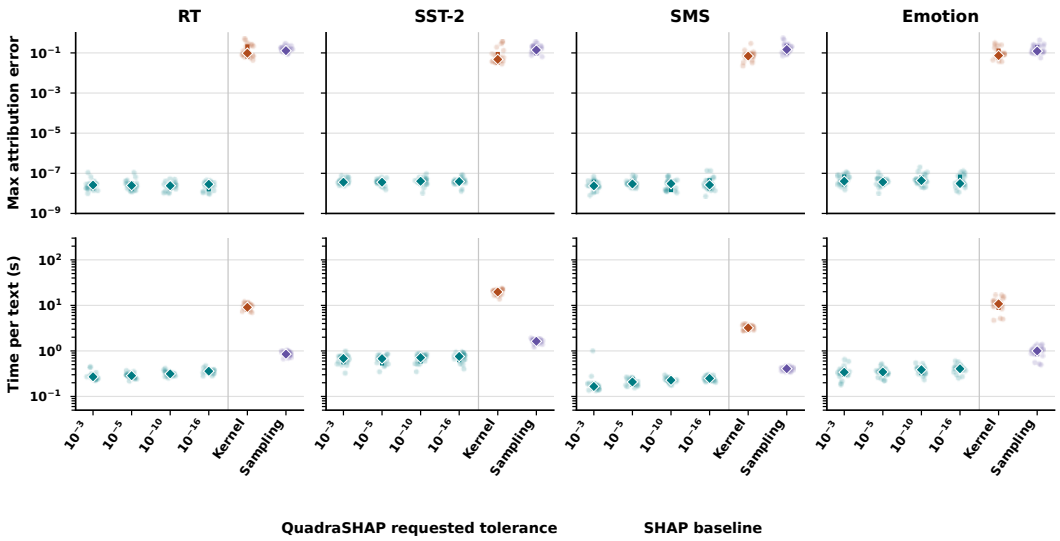

Figure 5: **Interventional attribution accuracy and runtime.** Top: maximum absolute attribution error per text against the float64 calculation. Bottom: seconds per explanation. All methods use the same 30-text background and 20 held-out inputs for each of four 5,000-feature RBF SVCs. QUADRASHAP runs on a GPU in float32 at four requested tolerances; KERNELSHAP and SAMPLINGSHAP run on CPU with `nsamples=1000`. Faint dots are individual texts, diamonds are medians, and vertical strokes span the 25th–75th percentiles.

far more than 1,000 samples per text for a reliable Shapley estimate, which made the comparison too time-consuming, so we restrict it to the other four corpora. Each corpus uses one fitted model; models, inputs, and backgrounds remain fixed across methods and tolerances.

QUADRASHAP runs with $\epsilon \in \{10^{-3}, 10^{-5}, 10^{-10}, 10^{-16}\}$. The implementations of KERNELSHAP and SAMPLINGSHAP use SHAP (Lundberg & Lee, 2017), the identity link, and `nsamples=1000`. Observed error is the largest absolute attribution difference across an input's features, measured against a separate float64 QUADRASHAP calculation of the same interventional value function at a tolerance below the rounding floor.

Accuracy and runtime. Figure 5 and Table 9 report the results. At $\epsilon = 10^{-5}$, QUADRASHAP uses five nodes (six on `sst2`), takes 0.21–0.68 s per text, and has median observed errors of 2.5–3.7×10^{-8}. KERNELSHAP takes 3.2–19.7 s with median errors of 0.05–0.10, and SAMPLINGSHAP 0.4–1.6 s with 0.12–0.15. These errors have to be read against the scale of the attributions: the largest attribution of a text has median magnitude 0.19–0.95 across the corpora, so the KERNELSHAP error is 10–43% of the largest attribution and the SAMPLINGSHAP error 16–84%, with relative ℓ_2 errors of 0.18–0.49 and 0.29–0.94. At this budget, neither estimator resolves the attributions; KERNELSHAP is the more accurate of the two on 61 of the 80 inputs. Therefore, QUADRASHAP is not only faster, but also provides a more accurate explanation compared to the sampling methods.

Table 9: Interventional comparison at selected tolerances, with 20 held-out texts and one fixed 30-text background per corpus. Columns report median node count, median time per text (seconds), and median and worst observed maximum-coordinate error against the float64 calculation. QUADRASHAP uses the float32 GPU backend; KERNELSHAP and SAMPLINGSHAP use CPU and `nsamples=1000`. The last column is the median of the error divided by the largest attribution of the same text. Dashes indicate that a quadrature node count is not applicable.

Corpus	Method	m_q	Time (s)	Median err.	Worst err.	Err. / $\max_j \lvert\phi_j\rvert$
RT	QUADRASHAP 10^{-3}	4	0.27	2.6×10^{-8}	1.1×10^{-7}	8.1×10^{-8}
	QUADRASHAP 10^{-5}	5	0.29	2.5×10^{-8}	1.1×10^{-7}	8.5×10^{-8}
	QUADRASHAP 10^{-10}	7	0.31	2.4×10^{-8}	5.4×10^{-8}	6.7×10^{-8}
	KERNELSHAP	–	9.07	0.097	0.442	0.26
	SAMPLINGSHAP	–	0.85	0.129	0.257	0.54
`sst2`	QUADRASHAP 10^{-3}	5	0.68	3.6×10^{-8}	7.4×10^{-8}	9.2×10^{-8}
	QUADRASHAP 10^{-5}	6	0.68	3.6×10^{-8}	6.2×10^{-8}	8.7×10^{-8}
	QUADRASHAP 10^{-10}	8	0.71	4.0×10^{-8}	9.2×10^{-8}	1.0×10^{-7}
	KERNELSHAP	–	19.72	0.048	0.311	0.12
	SAMPLINGSHAP	–	1.63	0.140	0.301	0.38
`sms spam`	QUADRASHAP 10^{-3}	4	0.17	2.4×10^{-8}	6.9×10^{-8}	1.6×10^{-7}
	QUADRASHAP 10^{-5}	5	0.21	3.0×10^{-8}	7.8×10^{-8}	1.6×10^{-7}
	QUADRASHAP 10^{-10}	7	0.23	3.1×10^{-8}	7.3×10^{-8}	1.5×10^{-7}
	KERNELSHAP	–	3.22	0.070	0.268	0.43
	SAMPLINGSHAP	–	0.41	0.147	0.492	0.85
`emotion`	QUADRASHAP 10^{-3}	4	0.34	4.0×10^{-8}	1.1×10^{-7}	5.5×10^{-8}
	QUADRASHAP 10^{-5}	5	0.35	3.7×10^{-8}	1.1×10^{-7}	5.5×10^{-8}
	QUADRASHAP 10^{-10}	7	0.39	4.4×10^{-8}	2.0×10^{-7}	6.4×10^{-8}
	KERNELSHAP	–	10.81	0.074	0.278	0.10
	SAMPLINGSHAP	–	1.00	0.122	0.377	0.16

G.5 TREE EXPLAINERS

The tree instantiation of Section F is benchmarked on the CPU against the established tree explainers (Section G.5.1) and on the GPU against Quadrature-TreeSHAP and GPUTreeSHAP, with TreeGradShap compared separately (Section G.5.2).

G.5.1 CPU BENCHMARK

We compare against SHAP (Lundberg & Lee, 2017), FASTTREESHAP V1/V2 (Yang, 2021), LINEAR TREESHAP (Yu et al., 2022a;b), and SHAPIQ (Muschalik et al., 2024). Numerical correctness is assessed via the worst-case violation of the Shapley efficiency axiom, which is zero by design for any exact implementation.

Synthetic data. We benchmark all tree-based methods on random forests trained on a controlled synthetic regression dataset (see Appendix G.2.1 for construction details), sweeping $d \in \{10, 100\}$ and ensemble size from 10 to 10^5 leaves. To assess algorithmic stability, we use $\max_i |\mathbb{E}[f] + \sum_j \phi_{ij} - f(x_i)|$ as a proxy, measuring the worst-case deviation from the efficiency axiom. Table 10 summarizes the order-of-magnitude differences in this stability proxy; QUADRASHAP ($m_q{=}\lceil \eta/2 \rceil$) is the fastest numerically stable method in every configuration. At 10^5 leaves it runs in 41.5 ms ($d{=}10$) and 48.3 ms ($d{=}100$), outperforming FASTTREESHAP V1 by 2–2.7× and SHAP by 3–5×, with the gap widening as the ensemble grows. Its efficiency-axiom violation remains at floating-point precision ($\sim 10^{-13}$) throughout. LINEAR TREESHAP matches QUADRASHAP in speed but is unreliable: its efficiency violation grows from $6{\times}10^{-14}$ at 10 leaves to $\sim$10 at 10^4 leaves ($d{=}10$), reaching $7{\times}10^2$ at 10^5 leaves ($d{=}100$). Instability sets in well before the asymptotic regime and worsens with tree depth, making its reported runtime meaningless at scale. SHAPIQ is correct but orders of magnitude slower, timing out at 10^5 leaves for $d{=}10$.

Real-world text classification. To assess performance on realistic, deep tree structures, we train random forest classifiers on five public text-classification datasets: `emotion` (Saravia et al., 2018),

Table 10: Tree explainer benchmark: runtime (milliseconds per instance) on synthetic data across leaf counts. Stability of the efficiency axiom $\max_i |\mathbb{E}[f] + \sum_j \phi_{ij} - f(x_i)|$: 🟢 exact ($<10^{-12}$), 🟠 degraded ($10^{-12}$ to 10^{-4}), 🔴 broken ($>10^{-4}$). Best per column **bold**; T/O denotes timeout after 300s.

	$d = 10$					$d = 100$				
#leaves	10	100	1k	10k	100k	10	100	1k	10k	100k
TreeSHAP	0.009 🟢	0.087 🟢	1.2 🟢	14 🟠	169 🟠	0.010 🟢	0.088 🟢	1.4 🟢	22.1 🟠	254 🟠
FastTreeSHAP v1	0.010 🟢	0.049 🟢	0.68 🟢	7.6 🟠	89.1 🟠	0.010 🟢	0.055 🟢	0.70 🟢	10.3 🟠	130 🟠
FastTreeSHAP v2	0.040 🟢	0.11 🟢	2.4 🟢	7.6 🟠	88.9 🟠	0.038 🟢	0.16 🟢	3.7 🟢	10.6 🟠	123 🟠
Linear TreeSHAP	0.015 🟢	0.050 🟠	0.48 🟠	6.1 🔴	69.1 🔴	0.016 🟢	0.050 🟠	0.49 🟠	5.9 🔴	66.5 🔴
SHAPIQ	3.3 🟢	53.8 🟠	573 🟠	6281 🟠	T/O	3.4 🟢	47.0 🟠	576 🟠	4602 🟠	46494 🔴
QuadraSHAP	**0.005** 🟢	**0.030** 🟢	**0.29** 🟢	**3.7** 🟢	**41.5** 🟢	**0.006** 🟢	**0.031** 🟢	**0.33** 🟢	**4.0** 🟢	**48.3** 🟢

Table 11: Tree explainer benchmark on five text-classification datasets: runtime (millisecond per instance) with stability of the efficiency axiom shown as 🟢 exact ($<10^{-12}$), 🟠 degraded (10^{-12} to 10^{-4}), 🔴 broken ($>10^{-4}$). Best per column **bold**; T/O denotes timeout after 300s.

	emotion	imdb	sms_spam	sst2	RT
max depth	100	20	100	100	100
total leaves	24,200	51,099	29,662	180,676	83,125
tree count	100	100	100	300	300
TreeSHAP	341 🔴	27.4 🟢	176 🔴	1707 🔴	773 🔴
FastTreeSHAP v1	326 🔴	18.4 🟢	166 🔴	1621 🔴	740 🔴
FastTreeSHAP v2	326 🔴	T/O	166 🔴	1618 🔴	740 🔴
Linear TreeSHAP	43.2 🔴	6.76 🔴	10.8 🔴	87.4 🔴	43.3 🔴
SHAPIQ	1650 🔴	2205 🟠	1930 🔴	T/O	T/O
QuadraSHAP	**11.3** 🟢	**3.75** 🟢	**9.68** 🟢	**60.5** 🟢	**27.3** 🟢

`imdb` (Maas et al., 2011), `sms spam` (Almeida et al., 2011), `sst2` (Wang et al., 2019), and `rotten tomatoes` (RT) (Pang & Lee, 2005). Documents are encoded as 5,000-dimensional TF-IDF vectors; full training, and dataset details are in Appendix G.2.3. The resulting ensembles range from 24k to 181k leaves and reach depths up to 100, substantially deeper than synthetic forests above.

Table 11 reports runtime and summarizes stability differences by order of magnitude. QuadraSHAP is the fastest method on every dataset, achieving 2–4× speedups over Linear TreeSHAP and 15–30× over SHAP and FastTreeSHAP. It is also the only method that remains numerically stable across all five datasets: at depth 100, every competitor exhibits catastrophic efficiency violations: SHAP and FastTreeSHAP reach $\sim 10^{9}$, SHAPIQ reaches $\sim 10^{30}$, and Linear TreeSHAP reaches 10^{186}–10^{271}, while QuadraSHAP stays at floating-point precision ($\leq 10^{-13}$). The exception is `imdb`, whose shallow forests (max depth 20) fall below the breakdown threshold, and all stable methods agree there.

G.5.2 GPU BENCHMARK

Experimental setup. We compared the CUDA implementation of QuadraSHAP with three closely related tree explainers: the GPU implementation of Quadrature-TreeSHAP in XGBoost (Wettenstein et al., 2026), the official `GPUTreeExplainer` of the SHAP library (Mitchell et al., 2022b), and the official TreeGrad-Shap implementation (Li et al., 2026). All methods computed first-order Shapley values under the path-dependent, cover-based tree value function of Section F.1. The primary GPU benchmark was conducted on an NVIDIA RTX 5090. We used the QuadraSHAP repository at commit `3aac1099ac26a59a42acca8f12fdfe6b320cafff`, XGBoost 3.4.1 and SHAP v0.52.0. The XGBoost binary was verified to use its Quadrature-TreeSHAP CUDA backend for `pred_contribs=True`. Only build-compatibility changes were required to compile SHAP for the available CUDA version; the GPUTreeSHAP algorithm was not modified. Synthetic

regression data were generated with target

$$y = \sum_{j=1}^{\min\{8,d\}} \left[\sin(x_j) + 0.35\, x_j^2\right] + \xi, \qquad \xi \sim \mathcal{N}(0,1).$$

The benchmark contained six model configurations: single trees with 16 features and 256, 4,096 or 16,384 leaves; a single tree with 100 features and 4,096 leaves; and forests of 32 trees with 128 leaves per tree and either 16 or 100 features. Tree depth was capped at 24 to remain within the path-size limitation of the released GPUTreeSHAP implementation, and every requested leaf count was attained. Each configuration was trained with three seeds (42, 43, 44) and evaluated at batch sizes $B \in \{1, 10, 64\}$, giving 18 model–batch cells. The same fitted tree topology, thresholds, leaf values, covers and input observations were supplied to all methods. Conversion to XGBoost introduced a maximum prediction discrepancy of 9.35×10^{-7}, primarily associated with XGBoost's single-precision tree representation.

Timing protocol. We measured the complete latency of each public explanation call rather than isolated CUDA-kernel time. Every timed call received a fresh contiguous host-array copy, and a fresh XGBoost `DMatrix` was constructed inside every Quadrature-TreeSHAP call; neither input containers nor previously computed explanations were cached. Model training, model conversion, explainer construction and initial CUDA compilation were excluded from the timed region. Every call was followed by explicit GPU synchronization, and numerical CPU libraries were restricted to one thread. For each seed we performed three warm-up calls followed by nine synchronized timed repetitions; the reported time for a model–batch cell is the median of the three seed-level medians and refers to the complete batch, not to an individual observation.

Table 12: GPU tree benchmark: representative median runtime in milliseconds for a complete batch of size B (lower is better).

Model	B	QUADRASHAP	Quadrature-TreeSHAP	GPUTreeSHAP
1 tree, $d = 16$, 256 leaves	1	**0.410**	1.327	2.407
1 tree, $d = 16$, 4,096 leaves	10	**1.564**	30.620	7.259
1 tree, $d = 16$, 16,384 leaves	10	**1.952**	121.789	22.307
1 tree, $d = 100$, 4,096 leaves	64	**3.156**	14.436	8.008
32 trees, $d = 16$, 128 leaves/tree	64	**1.426**	1.460	5.686
32 trees, $d = 100$, 128 leaves/tree	64	1.631	**1.544**	6.102

GPU results. The exact, structure-dependent QUADRASHAP implementation was faster than GPU Quadrature-TreeSHAP in 17 of the 18 model–batch cells. The geometric-mean speedup was $5.46\times$ and the largest speedup $62.38\times$, attained for the 16,384-leaf tree at batch size 10. The only Quadrature-TreeSHAP win occurred for the 32-tree, 100-feature forest at batch size 64, where it required 1.544 ms against 1.631 ms for QUADRASHAP, a $1.37\times$ advantage; the corresponding 16-feature forest was approximately tied at this batch size. Against GPUTreeSHAP, QUADRASHAP was faster in all 18 cells, with speedups from $2.48\times$ to $12.65\times$ and a geometric mean of $5.13\times$. The advantage increased notably for large single trees, where the quadrature formulation avoids the quadratic dependence on path depth of the original TreeSHAP recurrence.

Equal-node control. Because the XGBoost implementation of Quadrature-TreeSHAP uses a fixed budget of eight quadrature evaluations, we conducted a separate control in which QUADRASHAP was also forced to use $m_q = 8$. Both methods used their native eight-point rules, so the number of function evaluations was matched although the transformed nodes and weights were not identical. This experiment used five warm-ups and 25 synchronized repetitions per seed. Under this equal-budget condition QUADRASHAP remained faster in 16 of 18 cells, with a $5.03\times$ geometric-mean speedup and a maximum of $62.83\times$; Quadrature-TreeSHAP won the two batch-64 forest cases by $1.55\times$ and $1.59\times$. The forced-eight QUADRASHAP output differed from its exact output by at most 4.10×10^{-14} on this suite. The principal performance difference therefore cannot be explained by different quadrature-node counts.

TreeGrad-Shap comparison. The official TreeGrad-Shap implementation was evaluated separately on the same six model shapes, three seeds and three batch sizes, with one warm-up and five timed

repetitions. Since the released implementation is a single-row NumPy/Python CPU method, it is run on CPU, whereas QUADRASHAP ran on the RTX A6000; for forests, TreeGrad-Shap was applied to each estimator separately and the attributions were averaged, as linearity permits. QUADRASHAP was faster in all 18 cells, with speedups from $9.85\times$ to $3578.68\times$. These numbers compare the available implementations as complete systems and must not be read as an intrinsic algorithmic speedup, because devices, implementation languages and batching strategies differ substantially.

Numerical agreement. In an independent, untimed validation, TreeGrad-Shap and exact QUADRASHAP agreed to within 8.88×10^{-15}. The maximum attribution differences relative to exact QUADRASHAP were 1.79×10^{-6} for Quadrature-TreeSHAP and 7.26×10^{-6} for GPU-TreeSHAP, and exact QUADRASHAP satisfied the efficiency axiom to within 1.42×10^{-14}. The larger discrepancies of the external GPU implementations are consistent with their single- or mixed-precision representations and are not evidence that they compute a different path-dependent game.

Interpretation. The comparison with Quadrature-TreeSHAP does not establish a better asymptotic tree algorithm: both methods evaluate essentially the same quadrature representation. The observed difference is attributable to GPU execution and to the organization of the public call. QUADRASHAP prepares its compact tree representation once, retains it on the GPU, evaluates subtree quantities in levelwise bottom-up and top-down passes, and reduces contributions grouped by feature without atomic updates. The tested XGBoost path reconstructs and transfers its compressed quadrature representation during each public explanation call, schedules work by row and tree with a depth-first traversal, and accumulates contributions with atomic additions. The specialized CUDA tree implementation does not use the generic product-game evaluator of Section 3.3, so the measured tree speedup should not be attributed to it. For GPUTreeSHAP, both methodology and implementation contribute: quadrature removes the quadratic path-depth factor, while the released GPUTreeSHAP wrapper also repeats path construction, transfer, allocation and bin-packing work for each explanation call. The TreeGrad-Shap result is strictly a comparison of available implementations, as no GPU implementation of that method exists. Finally, these experiments used one GPU, synthetic regression forests, depths of at most 24 and batches of at most 64, and they time end-to-end public explanation calls rather than isolated kernels. They support a strong low-latency and large-single-tree result on the tested system, not a claim of unconditional dominance across hardware, datasets or workload shapes.